BIBLIOTECA BASICA INFORMATICA

¡Y LLEGO EL BASIC! (II) 7



INGELEK

BIBLIOTECA BASICA INFORMATICA

¡Y LLEGO EL BASIC! (II) 7

INGELEK

Director editor:
Antonio M. Ferrer Abelló.

Director de producción:
Vicente Robles.

Coordinador y supervisión técnica:
Enrique Monsalve.

Colaboradores:
Angel Segado.
Patricia Mordini.
Margarita Caffaratto.
Marina Caffaratto.
Francisco Ruiz.
Jorge Juan Monsalve.
Beatriz Tercero.
Fernando Ruiz.
Casimiro Zaragoza.

Diseño:
Bravo/Lofish.

Dibujos:
José Ochoa.

ISBN del tomo: 84-85831-38-1
ISBN de la obra: 84-85831-31-4
Fotocomposición: Pérez Díaz, S. A.
Imprime: Héroes, S. A.
Depósito Legal: M-37690-1985

INDICE

PROLOGO

En el primer volumen sobre el BASIC se describieron las instrucciones principales de este lenguaje, gracias a las cuales nos "aventuramos" a través de la escritura de algunos programas sencillos.

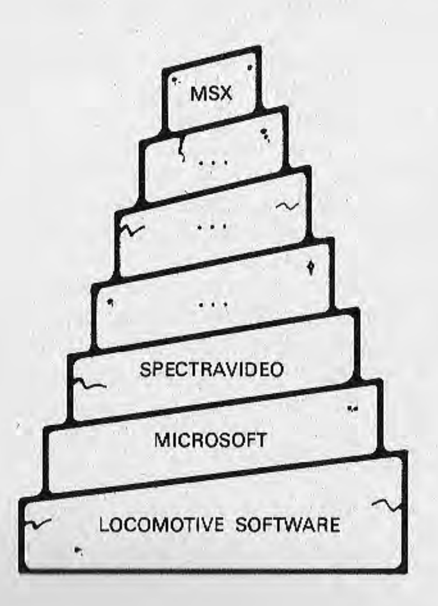
El BASIC es un lenguaje que parece estar concebido intencionadamente para los ordenadores personales, aunque sus orígenes se pierden en los primeros tiempos de las calculadoras electrónicas, cuando los ordenadores personales estaban todavía "por venir al mundo".

Los ordenadores personales (utilizamos el término personal como equivalente y sinónimo de ordenador "casero") son máquinas flexibles de las que podemos hacer el uso que más nos agrade. Hacia mitad de los años setenta fue cuando aparecían como tales los primeros. Inmediatamente se distinguieron de sus hermanos mayores, los grandes ordenadores tradicionales, por ser económicos, sencillos y, sobre todo, por no exigir un profundo dominio de la informática.

Hasta ese momento los ordenadores eran objetos muy exclusivos, utilizados sólo por grandes empresas o por centrales atómicas muy sofisticadas. Algunos años antes, la Hewlett Packard había utilizado el término personal para una de sus más características calculadoras de bolsillo, pero fue el Altair 8800 (vendido en kits de montaje por la MITS de Alburquerque, en Nuevo Méjico) quien abrió la era del ordenador "en casa", con todas las consecuencias sociales y culturales que de ello se han derivado. En el transcurso de muy poco tiempo nacieron el Apple I, hermano mayor del indiscutible líder Apple II, y el PET, que son los responsables de la noble dinastía de los Commodore.

El BASIC, inmortal como el fénix, renació precisamente gracias a su adopción como principal lenguaje de los ordenadores personales. Actualmente éstos, como coronación de un asombroso progreso tecnológico, pueden trabajar prácticamente con cualquier lenguaje de programación (FORTRAN, PASCAL, COBOL, C, FORTH, por citar los nombres más conocidos), pero su principal instrumento de programación sigue siendo el BASIC, que constituye un patrón y un indiscutible modelo de lenguaje para ordenador.

El entusiasmo por haber adoptado el simpático y fácil BASIC en los ordenadores personales, sin que existiera una versión oficial patrocinada por algún ente internacional, ha traído como consecuencia que cada fabricante de ordenadores personales o cada firma suministradora de software por encargo lo adaptara a sus propias exigencias, a menudo de carácter meramente comercial. La historia es vieja y ocurrió también con los ordenadores más grandes: tú compras mi hardware, pero solamente puedes utilizar mi software. Traicionar a un fabricante y pasarse a la competencia quería decir tirar a la basura el software y los programas de aplicación y volver a escribir todo a partir de la nada otra vez. Así sucedió con los ordenadores personales; cada uno puso a punto su propia versión del BASIC sin dejar ningún espacio al cliente para orientarse hacia otras máquinas.



Actualmente las cosas están cambiando. El mercado (¡el inefable mercado!) desea la compatibilidad entre programas, sistemas operativos y paquetes de software. Solamente quien respeta los convenios, tácitos o impuestos, sobre las normas (los estándares) supera las duras exigencias de los grandes mercados internacionales. Y así, mientras por una parte se afirma la versión del BASIC nacida en la prestigiosa firma americana Microsoft, de la que han surgido tantos vástagos de "sangre azul", por otra parte se están aclarando las posiciones de los sistemas operativos: los sistemas operativos CP/M y MS-DOS para los ordenadores personales, y el UNIX para la gama superior (por algunos llamada de los "micro" y que tiempo atrás abarcaba a los famosos "miniordenadores"). Pero hay quien no se quiere someter a ninguna normalización y la "torre de Babel" crece otro piso más. Y así llegamos a un nuevo pretendiente a la corona de unificador: el sistema MSX.

El MSX nació también en la casa Microsoft, pero inmediatamente fue adoptado más allá de los océanos por los fabricantes del Sol Naciente, en primera instancia, y por algunos europeos después. Buscaba contrastar el predominio comercial americano y propone una unificación que, por primera vez, implica no solamente al software, sino también al hardware.

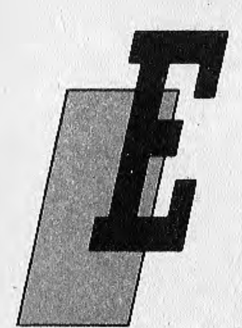
Las buenas intenciones de los fabricantes que adoptan el sistema MSX están a favor del usuario: todos los programas escritos en los ordenadores MSX son transferibles a cualquier otra máquina que adopte este sistema. Y también, todos los periféricos que adoptan el patrón de conexión MSX se pueden conectar a cualquier otro ordenador que lo utilice. Ya no tendremos necesidad, por ejemplo, de joysticks diferentes para cada ordenador, sino que cada joystick servirá para cualquier máquina.

Los fabricantes podrían así dedicarse a mejorar el producto sabiendo que el mercado potencial se hace cada vez más amplio. Lo mismo es aplicable a los discos flexibles y a los cartuchos de memoria ROM o a las impresoras. Es una cuestión de normalización que ya se planteó otras veces, pero que en muy raras ocasiones tuvo éxito. Solamente los grandes intereses comerciales e industriales pueden imponer una normalización, y en un próximo futuro veremos si el sistema MSX, u otros, son capaces de imponerse definitivamente. Nosotros, los usuarios del ordenador personal, no esperamos otra cosa sino que estos acuerdos se hagan una realidad tangible.

CAPITULO I

PERIFERICOS Y CONEXIONES

De qué BASIC hablamos



En la parte introductoria de la primera monografía dijimos que hablar de BASIC en general quiere decir describir su estructura, presentar sus instrucciones principales, los conceptos que son comunes a todos los dialectos y advertir al lector cuando haya diferencias importantes.

También dijimos que trataremos de hablar de BASIC en los términos más comunes posibles y que las más importantes diferencias entre versiones se encuentran en las instrucciones de gráficos (dibujos en la pantalla) o de gestión de ficheros (los archivos de datos), instrucciones que pretendemos presentar en esta monografía. Para establecer contacto con lo expuesto en el anterior volumen recordamos que las instrucciones de un lenguaje de programación se pueden dividir, a grandes rasgos, en cuatro familias:

- las instrucciones declarativas,
- las instrucciones de entrada/salida,
- las instrucciones de cálculo y asignación,
- las instrucciones de control.

Esta subdivisión no es solamente conceptual, pues existe siempre una correspondencia física con un elemento diferente del hardware del ordenador. Por ejemplo, las instrucciones de entrada/salida se relacionan con la gestión o control de las unidades periféricas (impresoras, discos, etc.), mientras que las de asigna-

ción se refieren a la atribución de valores a las celdillas de la memoria central.

Las instrucciones que nos proponemos estudiar en esta monografía pueden incluirse en estas familias, pero exigen una mayor atención y, con frecuencia, una descripción más detallada que las vistas en el volumen anterior. Se trata de instrucciones que solamente deben ser empleadas cuando se haya adquirido un buen dominio de las demás (las descritas en la primera monografía) y cuando, frente a un error de sintaxis indicado en la pantalla, se tengan los conocimientos y la buena voluntad de analizar el programa y detectar el error.

Así, hablaremos de GOSUB, que es la instrucción del BASIC para llamar a una subrutina. GOSUB puede clasificarse entre las instrucciones de control, pero merece una posición privilegiada porque permite dar una cierta estructuración a los programas, como veremos dentro de poco.

Haremos también alusión a algunas instrucciones particulares de asignación que, por falta de espacio, no se presentaron en la monografía anterior: RND y DEF FN. La primera, de gran utilidad, permite generar números al azar, mientras que la segunda pone a disposición del programador la posibilidad de construir funciones (en sentido matemático) nuevas con respecto a las ya proporcionadas por el lenguaje BASIC.

Abordaremos luego uno de los temas más complicados, el "nivel universitario" del BASIC, que es la gestión de los archivos o ficheros, como se dice en la jerga informática.

Las instrucciones sobre ficheros adolecen muchísimo de las derivaciones propias de las distintas versiones, por lo que trataremos de hablar mucho de conceptos y dar ejemplos que se refieran solamente a los ordenadores de mayor difusión. Nos excusamos, desde ahora mismo, por no tratar las instrucciones de todos los ordenadores existentes en el mercado, pero ello sería verdaderamente imposible. Lo que sí haremos, como en el volumen anterior, es presentar una "tabla de equivalencias" de las instrucciones en distintos ordenadores.

Por último, veremos cómo se puede dibujar con el ordenador, y dentro de este tema abordaremos los gráficos. También en este caso, y casi en mayor medida, se pueden describir muchos conceptos, pero daremos algunos ejemplos que no sean específicos, en la medida de lo posible, de una sola máquina. Por desgracia, si con los ficheros se pueden describir situaciones que abarcan a muchos ordenadores similares, con los gráficos cada ordenador es un mundo aparte.

Con frecuencia, incluso modelos de ordenadores de una misma firma se comportan, en lo que respecta a los gráficos, de un modo absolutamente diferente. No hay más remedio que asimilar los conceptos básicos (lo que nunca vendrá mal) y luego, aplicarlos a los casos concretos.

Antes de comenzar a hablar de instrucciones del BASIC, y teniendo en cuenta que esta monografía trata muchas instrucciones de entrada y de salida, consideramos oportuno hacer un examen de las unidades periféricas más importantes. A menudo, los problemas de programación no son conceptuales ni teóricos, sino que se derivan del desconocimiento real de las unidades periféricas.

El ordenador y sus periféricos

Nos hemos propuesto hablar solamente del BASIC y, por consiguiente, de software, pero no es posible prescindir de una breve descripción de las unidades periféricas del ordenador si queremos comprender mejor el significado y funcionamiento de las instrucciones que tratan de la entrada y de la salida de los datos.

Un ordenador procesa datos, pero para que ello sea posible es preciso que desde el exterior dichos datos se le suministren junto con los programas que los deben tratar. Estas funciones de comunicación con el exterior no son competencia, hablando en términos estrictos, del ordenador propiamente dicho (o mejor dicho, de la CPU), sino que se desarrollan mediante dispositivos adecuados, llamados unidades periféricas, o de una forma más sencilla, periféricos. Así, cuando se quiere describir de una manera concisa un sistema basado en ordenador, se dibuja la unidad central (la CPU) y alrededor de ella todos sus periféricos (Fig. 1).

Los periféricos son, por ejemplo, las impresoras, las unidades para discos, los digitalizadores y los joysticks, pero también el teclado y el monitor son periféricos.

Alguien podría considerar que los periféricos tienen una importancia secundaria en relación con el ordenador. Esto no es cierto en absoluto y, por el contrario, con mucha frecuencia la capacidad de un ordenador depende, en gran medida, de los periféricos de que dispone. Basta un ejemplo para constatarlo: resulta inútil que un ordenador sea capaz de calcular millares de datos por segundo si luego su impresora sólo puede escribir diez o cien por segundo.

El primer cometido solicitado a los periféricos consiste en permitir la comunicación, en uno u otro sentido, entre el ordenador y el hombre.

Dentro de los periféricos de entrada (aquellos que comunican al hombre con el ordenador) se encuentran:

- teclados,
- ratón,

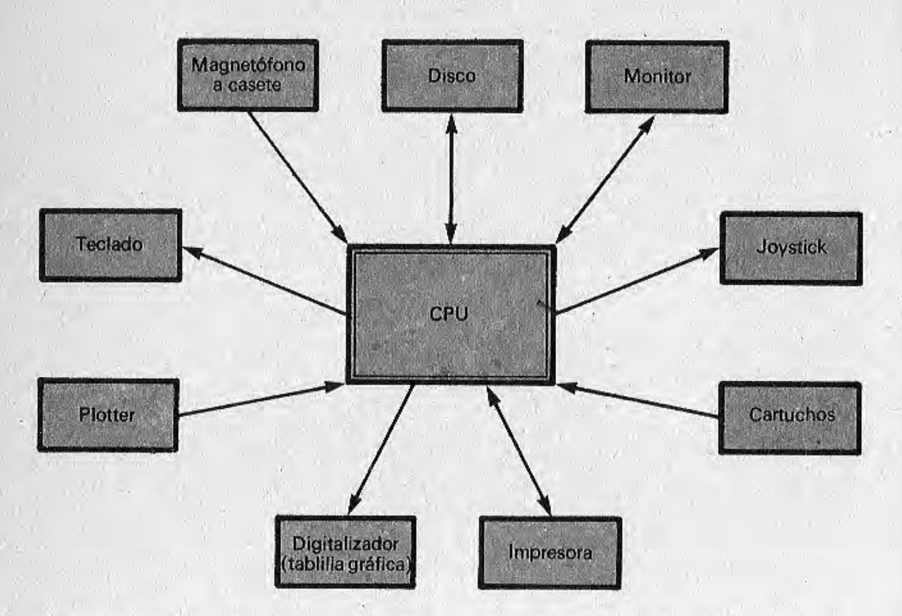


Figura 1.—Diagrama de bloques de un sistema de ordenador, con la unidad central de proceso (CPU) en el centro y los periféricos alrededor.

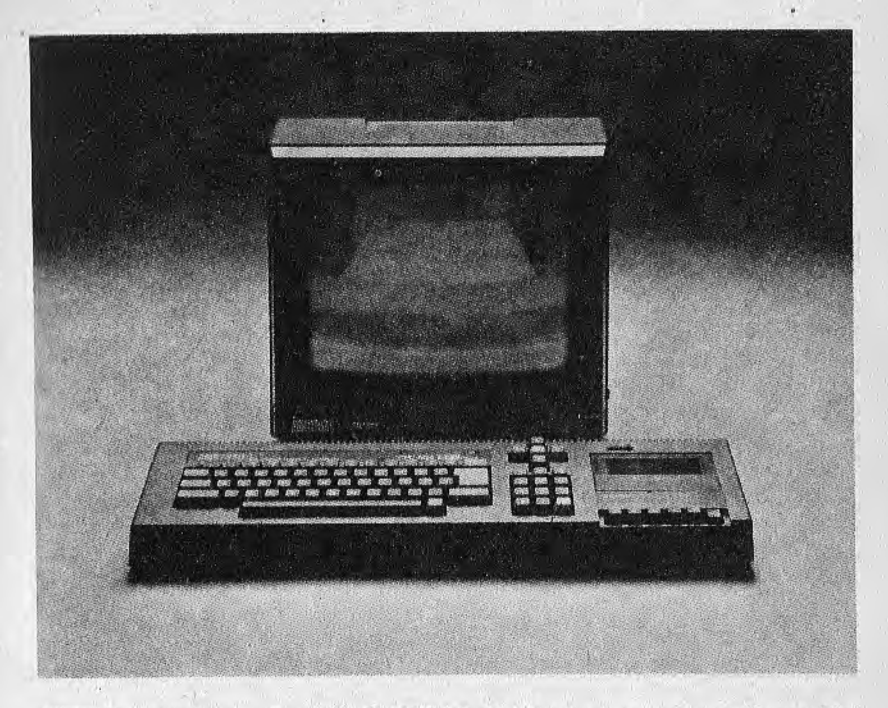
- digitalizador (tablilla gráfica),
- lápiz óptico ("ligh pen"),
- paddle ("paletas"),
- joystick,

y entre los periféricos de salida (envían mensajes del ordenador):

- pantalla de presentación visual (o monitor),
- impresoras,
- plotter (trazador).

La lista anterior no es completa, por supuesto, pero el concepto está claro: todos ellos son dispositivos, más o menos complejos, que permiten proporcionar informaciones al ordenador (escribiendo, dibujando, etc.) o recibirlas de él (en pantalla, en papel).

Además de todos los anteriores hay periféricos que intercambian datos solamente con la CPU y en los cuales el hombre no interviene de manera directa. A grandes rasgos, se pueden dividir en dos grupos: los periféricos de almacenamiento masivo y los dispositivos de telecomunicación. El primer grupo comprende:



Fotografía 1.—Equipo AMSTRAD-464. Incluye en su configuración básica teclado, monitor y magnetófono a casete.

- cinta magnética,
- casete,
- discos flexibles (disquetes),
- discos rígidos.

El segundo grupo está caracterizado por un único dispositivo significativo. Dicho dispositivo es el modem que permite modular una señal para su transmisión y, luego, demodularla (MOdulador/DEModulador).

Periféricos de entrada/salida menos conocidos

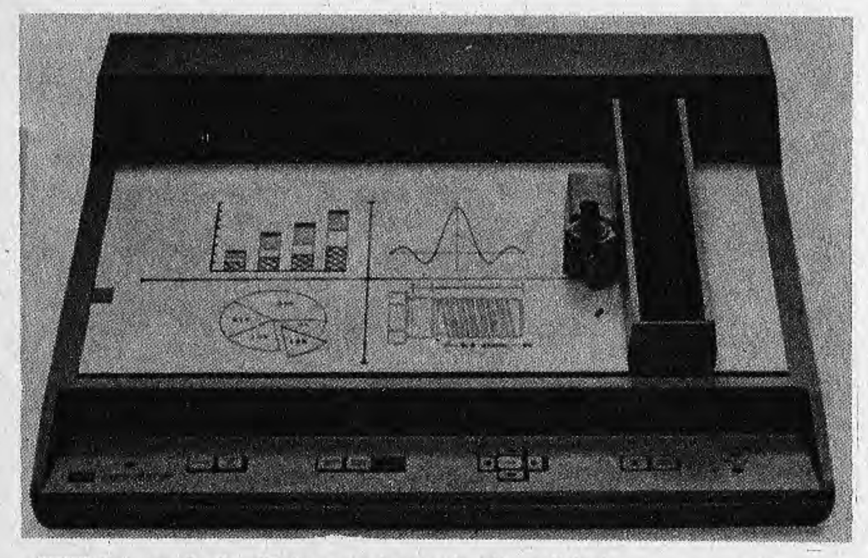
El ratón se asemeja a un paquete de cigarrillos con un pulsador y en su parte inferior, una bola "aprisionada". Al moverlo con la mano sobre el escritorio se desplaza en correspondencia un cursor indicador en la pantalla. De este modo resulta posible impartir órdenes al ordenador de una manera rápida, fácil y sobre

todo sin conocer nada de programación. En el espíritu de sus creadores se trata de una alternativa a los teclados tradicionales.



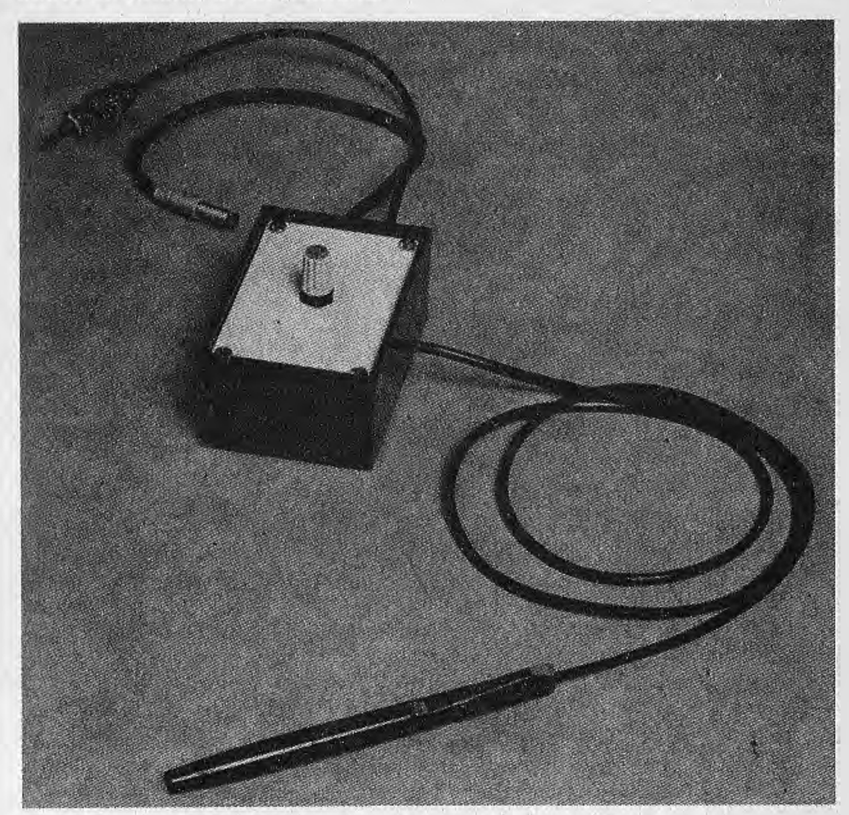
Fotografía 2.—Caja del ratón, con su "rabo" característico.

El digitalizador (tablilla gráfica) está constituido por una especie de pizarra y por un lápiz; al desplazar el lápiz sobre la pizarra se envía al ordenador la forma digital de un dibujo.



Fotografía 3.—Plotter moderno de alta calidad.

El lápiz óptico funciona de modo análogo, pero se trata de un lápiz provisto de una célula fotoeléctrica que se tiene que apoyar sobre el cristal de la pantalla.



Fotografía 4.—Periférico muy reciente: el lápiz óptico.

El joystick está provisto de una palanca móvil y a veces de un pulsador, a diferencia del paddle, que emplea una manilla giratoria.

Los plotters efectúan trazados sobre hojas de papel, sobre una superficie plana o sobre un cilindro y pueden utilizar varios lápices coloreados.

Como dijimos anteriormente, muchas veces la velocidad de los periféricos que interaccionan con el hombre no es un factor determinante para el buen funcionamiento del ordenador. Por ejemplo, es inútil que una pantalla sea capaz de visualizar millares de caracteres por segundo, puesto que nosotros nunca los po-



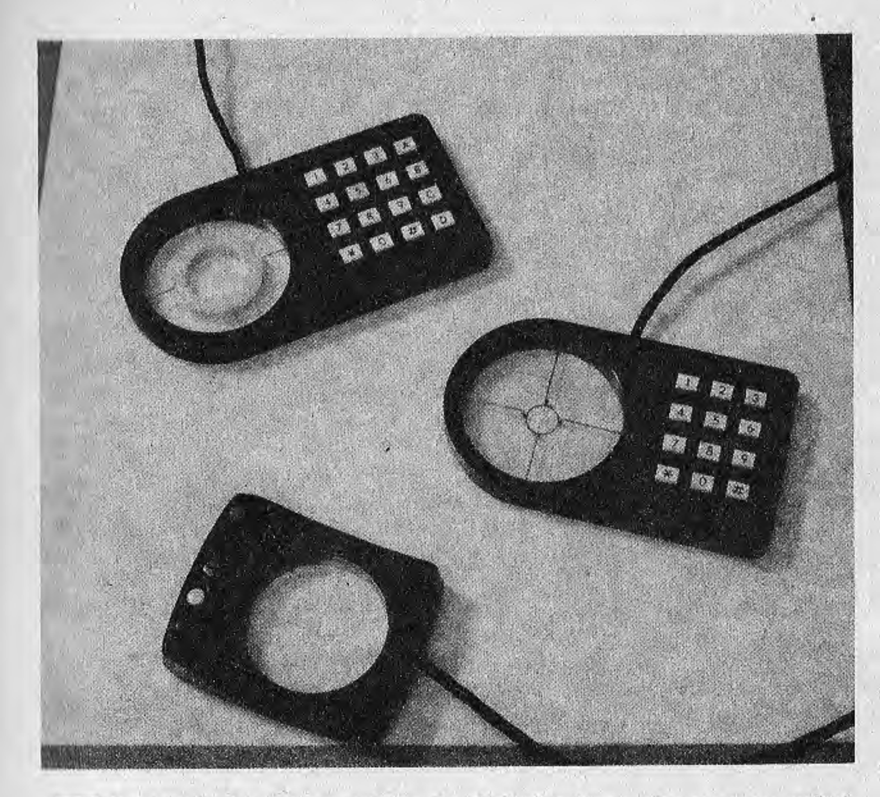
Fotografía 5.—Diversos joysticks.

dríamos leer. Por el contrario, una impresora demasiado lenta puede plantear problemas cuando se tengan que imprimir muchos formularios.

La cuestión es diferente para los periféricos que comunican solamente con el ordenador, tales como los discos flexibles. En tal caso, siempre se busca alcanzar la máxima velocidad posible.

Periféricos de almacenamiento masivo

Para el usuario humano, los programas y los datos son cosas conceptualmente muy distintas: los primeros son las instrucciones



Fotografía 6.—Tres modelos de digitalizadores, alguno muy sofisticado.

necesarias para procesar a los segundos. Para el ordenador, en cambio, ambos son secuencias de números binarios (hileras largas de ceros y unos) conservados en la memoria central.

Lamentablemente, la memoria central, aunque cada día más barata, es todavía bastante costosa y, por consiguiente, su amplitud total nunca es demasiado "generosa". Además, las memorias que utilizamos en la actualidad (de coste relativamente moderado) son siempre volátiles y pierden su contenido al apagar el ordenador. Por estos dos motivos resulta muy importante que los programas y los datos se conserven en memorias no volátiles.

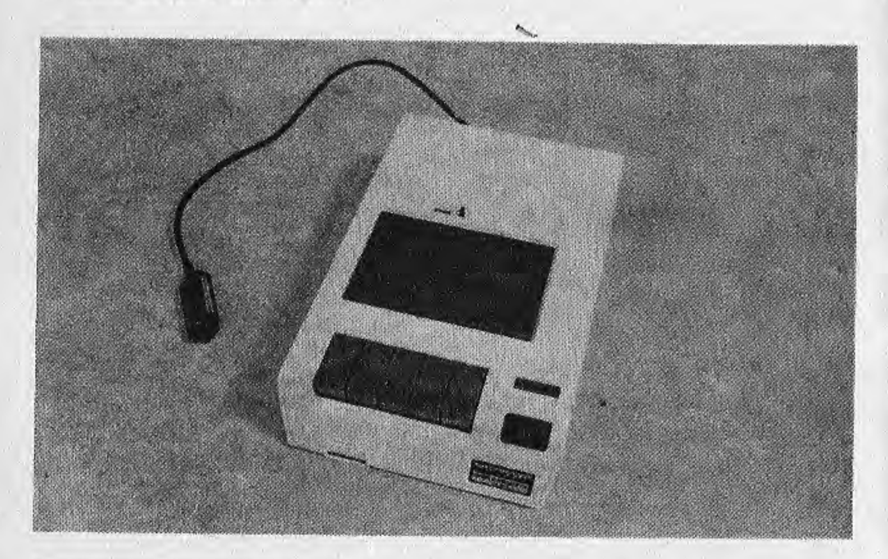
Los periféricos de almacenamiento masivo, denominados también memorias de masa, porque pueden contener grandes cantidades de información, son precisamente de tipo no volátil. Entre los más importantes están:

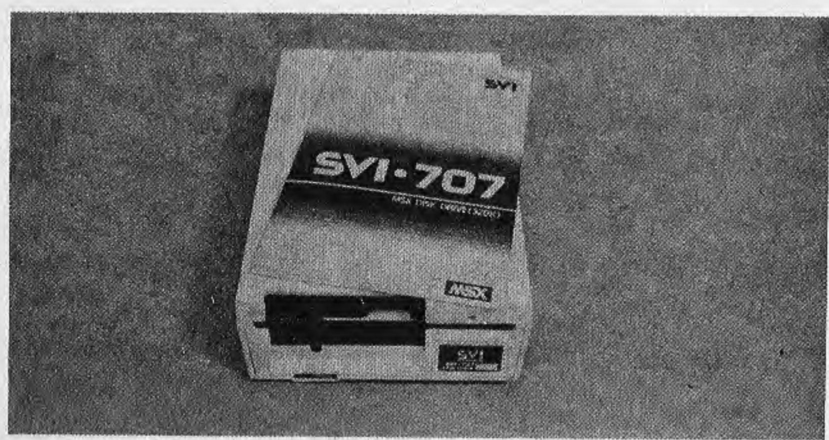
- cintas magnéticas,
- tambores magnéticos,

- cintas de casete magnéticas,
- discos magnéticos rígidos ("hard disk"),
- discos magnéticos flexibles ("floppy disk"),

y además:

- unidad de "burbujas magnéticas",
- unidad de disco óptico.





Fotografía 7.—Periféricos de almacenamiento masivo más usados: casete y disco flexible.

Como se observará, la tecnología actualmente más difundida es la magnética, en la que cada bit se almacena bajo la forma de un campo magnético microscópico en un soporte constituido por óxidos de hierro depositados sobre una base de mylar (cintas, casetes y disquetes) o de aluminio (tambores y discos rígidos). Esta tecnología es muy similar a la empleada para las grabaciones musicales ordinarias, y tanto es así que se suelen utilizar los mismos magnetófonos a casete de estas últimas con los ordenadores, incluso para grabar datos y programas.

Un parámetro muy importante en las memorias de masa es el tiempo de acceso a los datos, es decir, el tiempo necesario para acceder a una información determinada. Los discos rígidos, que giran continuamente a gran velociad, tienen tiempos de acceso muy pequeños; las cintas magnéticas y las cintas de casete, por el contrario, son muy lentas por cuanto que es preciso desarrollar toda la cinta que precede al punto buscado, esto es, se trata de dispositivos

Conexiones serie y paralelo

Para comprender mejor la forma en que trabajan los periféricos con el ordenador y, por consiguiente, para saber cómo deben escribirse las instrucciones de entrada y de salida de datos en los programas, es útil aclarar un par de conceptos de hardware. A tal fin hablaremos de conexiones en serie y en paralelo entre dos dispositivos.

En las conexiones en paralelo (Fig. 2) hay un hilo eléctrico por cada bit objeto de transmisión. Si se tienen que transferir 8 bits existirán, pues, 8 hilos (más el de masa, claro). Cada bit se transmite como un breve impulso eléctrico. De forma análoga, habrá tantos hilos como bits, incluso en los ordenadores más grandes de 16 o de 32 bits, si se conectan en paralelo a cualquier periférico.

En el caso de conexiones en serie (Fig. 3), por el contrario, los bits se envían uno a uno a lo largo de un solo hilo (más el de masa) a intervalos de tiempo periódicos. La transmisión de datos en serie se utiliza para conexiones económicas o en donde no hay alternativas (por ejemplo, para poder explotar las líneas telefónicas). Este sistema es muy similar a la comunicación telegráfica en código morse y es prácticamente la misma utilizada por los teletipos ordinarios.

Los buffers o memorias tampón

Un elemento importante, cuando se habla de intercambios de datos entre dos dispositivos, es el buffer o memoria tampón. He-

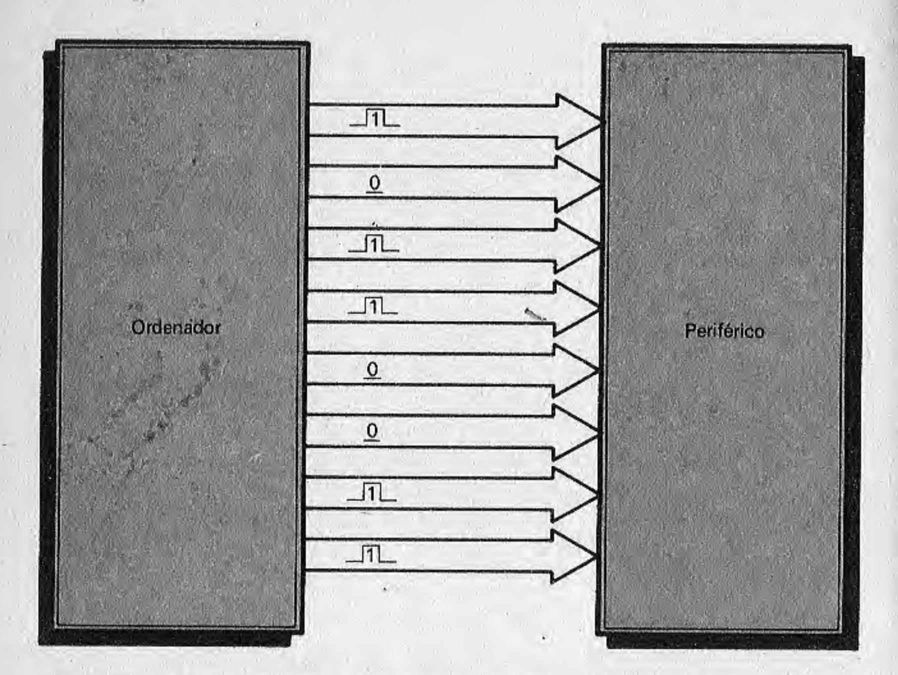


Figura 2.—Conexión en paralelo: una línea para cada bit.

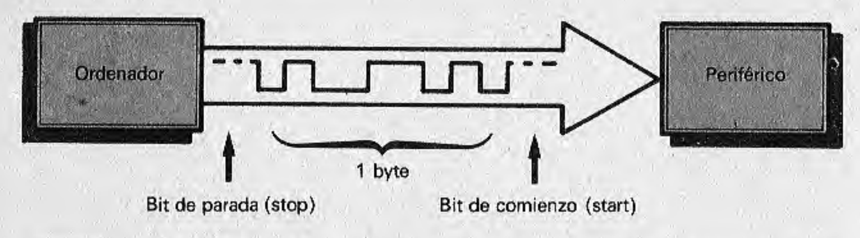


Figura 3.—Conexión en serie: por una sola línea se transmiten sucesivamente todos los bits.

mos supuesto, al hablar de conexiones en serie y en paralelo, que los dispositivos transmisores y receptores trabajan a la misma velocidad. Por ejemplo, si el ordenador transmite 2 Kbytes por segundo (2.000 bytes por segundo), el periférico debe poder recibir a la misma velocidad.

No siempre es posible esta igualdad de velocidad. Los circuitos electrónicos pueden trabajar con gran rapidez (tiempos del orden de magnitud de microsegundos o incluso inferiores), mientras que los dispositivos con partes mecánicas son mucho más lentos. Si suponemos que estos últimos trabajan a una velocidad de milisegundos, que es muy elevada para su propia naturaleza, seguirán siendo mil veces más lentos que los dispositivos electrónicos (en un microsegundo hay mil milisegundos).

Por ejemplo, una unidad de disco no puede tener acceso a un dato hasta que el giro del disco no lleve el dato deseado bajo la cabeza de lectura/escritura. Si el ordenador tiene que escribir 100 datos en un disco sería inadmisible tener que esperar cada vez una vuelta completa del disco (que suele tardar 200 milisegundos por vuelta) y, por consiguiente, unos 20 segundos en total.

Se recurre entonces a la técnica de la memoria buffer. Buffer significa memoria intermedia. Actúa como una especie de amortiguador, al igual, por ejemplo, que los topes de los vagones de ferrocarril. También la batería del automóvil es un ejemplo de buffer. En nuestro caso, sin embargo, los buffers no amortiguan acumulando energía, sino datos.

Los datos producidos por la CPU se acumulan en el buffer, que no es otra cosa que una pequeña zona de memoria RAM, sin hacer más lento el trabajo de la propia CPU (Fig. 4). Cuando el buffer está lleno, se descarga enviando sus datos hacia el periférico (a veces, incluso, sin necesidad de que intervenga la CPU). En el ejemplo de la unidad de disco, el tiempo necesario para la operación se reduce así a una sola vuelta del disco.

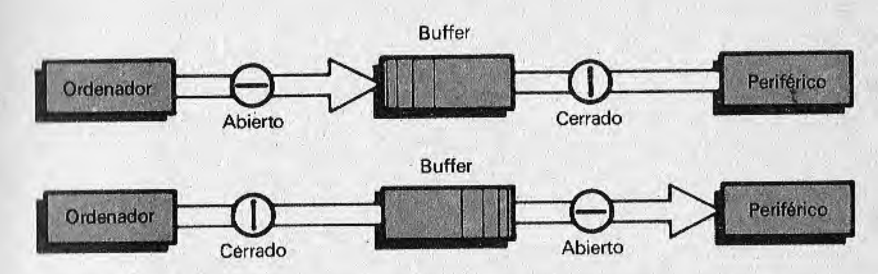
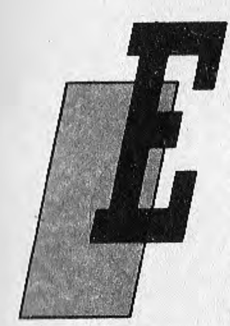


Figura 4.—Un buffer trabaja como un depósito de datos intermedio.

A veces, el buffer está incorporado en el propio periférico, como en el caso de algunas impresoras que admiten un millar de bytes a la velocidad máxima con que los produce la CPU; los acumulan en una memoria buffer interna y luego "con toda calma" los imprimen a la velocidad permitida por la mecánica. Así, mientras tanto, la CPU puede desempeñar otras tareas.

CAPITULO II

SUBRUTINAS Y DEFINICION DE FUNCIONES



Escribir programas en BASIC, como en cualquier otro lenguaje de programación, puede ser fácil y agradable. A menudo se obtienen resultados satisfactorios incluso con programas solamente de algunas decenas de instrucciones, pero cuando se escriben programas complejos profesionales, pueden ser necesarias centenares o millares de líneas de programa. En este caso, el modo en el que se escribe el programa, es decir, su estructura como la llaman los expertos, se hace fundamental tanto para la corrección de su funcionamiento como para su legibilidad. La legibilidad de un programa consiste en que cualquiera que observe su listado pueda comprender cuál es el principio del funcionamiento, o dicho de otro modo, el algoritmo base del propio programa. De la relación entre programas y algoritmos hablamos ya ampliamente en la primera monografía sobre el BASIC.

El BASIC no es un lenguaje adaptado al tipo de programación que los expertos denominan estructurada, como es el caso del lenguaje PASCAL o del más moderno ADA. ¿En qué consiste ésta? Intentar aclarar esta materia, incluso para los programadores de BASIC, nunca viene mal.

Dar una definición de programación estructurada nos podría llevar mucho tiempo y, dado el carácter divulgativo de estas monografías, nos complicaría las ideas en lugar de aclararlas. Digamos que un programa que tenga intenciones estructuralistas es un programa escrito y desarrollado con claridad que puede ser comprendido por cualquiera que conozca el lenguaje de programación que utiliza. Con demasiada frecuencia vemos programas que no son comprensibles ni siquiera para personas muy versa-

das en BASIC. En ocasiones hasta el mismo autor, transcurrido algún tiempo, no llega a comprender qué significa su listado, y si debe buscar un error o realizar una modificación "a tientas" (son los denominados "programas laberintos").

¿Cómo es posible que suceda esto? Basta con utilizar muchas instrucciones GOTO o emplear el mismo nombre de variable en varios puntos del programa con significados diferentes. Un programa es un proceso lógico que debe tener un comienzo y un final, y cada disgresión debe señalarse con claridad. Si un determinado grupo de instrucciones tiene un cometido concreto pongamos delante una instrucción de comentario (REM) para aclarar dicho cometido, o bien utilicemos los dos puntos (:) después de un número de línea (u otro carácter permitido por el BASIC) para separar grupos de instrucciones y dar un aspecto arquitectónico al programa. Veamos un ejemplo:

```
100 REM PROGRAMA PARA EL CALCULO
110 REM DEL PASO
120 REM DEL COMETA HALLEY
130 REM
140 REM
150 REM ULTIMO PASO DEL COMETA HALLEY
160 REM
170 REM
180 REM CALCULO DE LA ORBITA
190 REM
200 REM
210 REM
220 REM PERIHELIO EL DIA 9 DE FEBRERO DE 1986
230 REM
240 REM
250 REM PROXIMO PASO EN EL AÑO 2062
```

Pero supongamos ahora que queremos desarrollar un nuevo programa. Tendremos que proceder con mucho orden. Tuvimos ya ocasión de decir que la peor manera de realizar un programa es la de sentarse inmediatamente delante del ordenador y comenzar a escribir de golpe las instrucciones como si se tratase de desahogar un impulso de inspiración literaria. Ante todo, tenemos que escribir en un papel los pasos principales del programa (o sea, del algoritmo) con ideas muy claras sobre lo que queremos; y luego partiremos de esta descripción para desarrollar la estructura del programa.

Veamos un ejemplo común para muchos casos prácticos, como es la escritura en un fichero (archivo) de nombres recibidos desde el teclado. Los pasos del programa son los siguientes:

1. Enviar a la pantalla los mensajes de aviso al usuario iniciales.
2. Pedir un nombre.

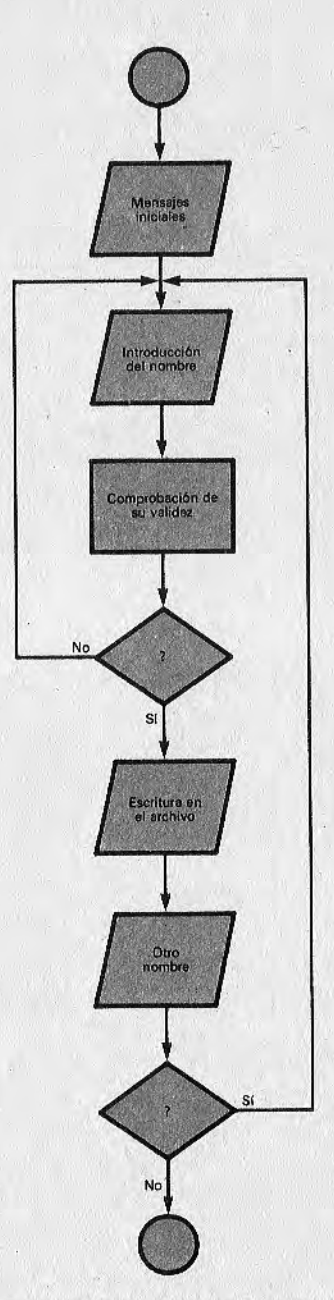


Figura 1.—Diagrama de bloques explicativo, a partir del cual se escribirá el programa.

3. Comprobar la validez formal del nombre.
4. Si es correcto, escribir el nombre en el archivo (disco flexible). En caso contrario, volver a pedirlo (2).
5. Preguntar si hay más nombres que introducir y, si fuera así, volver al punto 2; en caso contrario, terminar el programa.

Antes de proseguir, dibujemos el diagrama de bloques de nuestro programa, tal como lo hemos definido hasta ahora (Fig. 1).

Para facilitar esto, entre otras cosas están las subrutinas. Una subrutina es, en esencia, una serie de instrucciones, separadas de la secuencia principal del programa, que cumplen una misión específica. Tiene un punto de comienzo y uno de final propios. Cuando se la quiere usar desde el programa "principal" se hace un salto especial al punto de comienzo y, al llegar alfinal, la subrutina hace otro salto a la instrucción siguiente en el programa principal, a aquella por la que fue "llamada". El salto a la subrutina se realiza con la instrucción GOSUB. (Fig. 2).

Si al escribir el programa logramos mantener clara la división entre los cinco puntos anteriores, habremos hecho una buena labor.

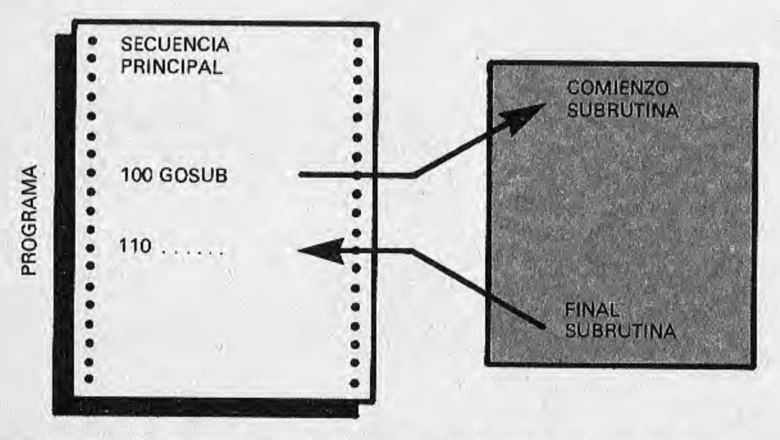


Figura 2.—Forma de emplear una subrutina desde el "programa principal".

GOSUB

Si se fija en el punto 3, relativo a la comprobación de la validez de los datos recibidos a la entrada, observará que se trata de

algo separado, desde el punto de vista conceptual, del núcleo del programa. El punto 3 podría "confiarse" a una subrutina de servicio, que se ocupe exclusivamente de efectuar las comprobaciones, y a la que acudiríamos cada vez que necesitásemos efectuar una.

Pensándolo bien, también el punto 4 puede llegar a ser una subrutina. En realidad, se ocupa solamente de llevar a un archivo los nombres. En esta primera escritura del programa hemos considerado oportuno grabar el archivo en un disco flexible, pero podríamos querer hacerlo también en una cinta de casete. Si utilizamos una subrutina será muy fácil cambiar solamente las instrucciones necesarias para pasar del disco a la cinta de casete, mientras que, si no la usamos tendríamos que cambiar el programa principal, y ello resultaría más complicado.

El empleo de subrutinas, que dependen de un programa principal, nos permite imaginar la estructura de bloques de un programa de una manera diferente y más flexible. En la Figura 3 se ilustra la forma en que se modifica el diagrama de bloques anterior al emplear subrutinas.

Veamos ahora cómo se utiliza la instrucción GOSUB para llamar a las subrutinas en BASIC. En la jerga informática se dice llamar una subrutina, porque el FORTRAN, el primer lenguaje que utilizó las subrutinas, tiene una instrución CALL (llamar) que sirve precisamente para activar la ejecución de las subrutinas. En la práctica se dice que el programa principal llama a las subrutinas. Siempre por motivos históricos se habla también en BASIC de paso de los parámetros para indicar el intercambio de datos y variables entre el programa principal y las subrutinas, pues en FORTRAN antes de poder emplear una subrutina es preciso "comunicarle" sobre qué parámetros o argumentos tendrá que actuar. En el BASIC esto no es estrictamente necesario porque las subrutinas no son programas totalmente separados, sino que forman un todo junto con el propio programa principal, compartiendo la memoria del ordenador, con lo que todas las variables son comunes.

Un ejemplo de subrutina

Para ver como se utilizan las subrutinas, tratemos de escribir un programa que haga uso de ellas.

Un caso muy frecuente cuando se trabaja con los ordenadores es el de que se tenga que dar una fecha nada más conectarlo. Un ordenador no está obligado, a no ser que lo programemos para ello, a comprobar lo que le comunicamos, y así, si quisiéramos bromear y dar fechas como el día 66 del mes 25 del año 234, lo podríamos hacer con "toda impunidad".

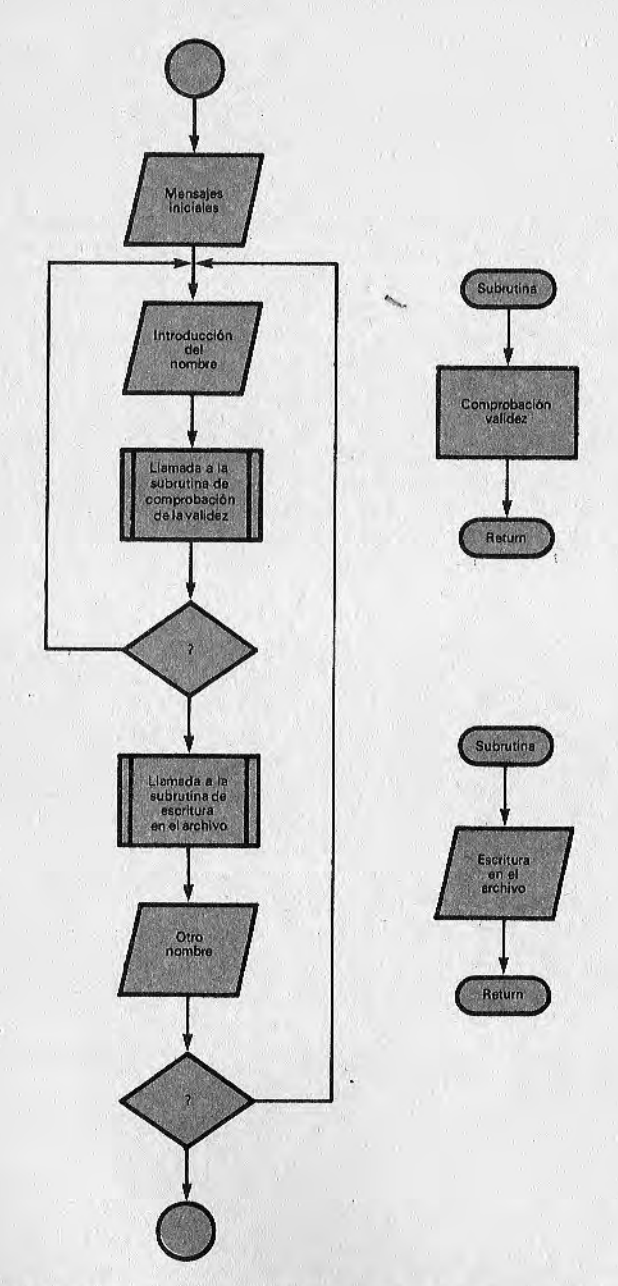


Figura 3.—Diagrama de la figura 1 modificado por el empleo de subrutinas.

En cambio, es muy fácil que al menos la corrección formal de una fecha sea controlada; el día del mes debe estar comprendido entre los valores establecidos, el número del mes no debe ser superior a 12, etc. Por consiguiente, si en un programa del tipo de control de un registro civil tenemos que pedir muchas veces las fechas, merece la pena dedicar algún tiempo a establecer una subrutina de control, llamada inmediatamente después de que sea introducida una fecha, y que compruebe su validez.

Supongamos que en varios puntos de un programa hay instrucciones, como las líneas 100 y 110 del ejemplo siguiente, en las cuales se visualiza un mensaje de aviso: "dar la fecha como día, mes y año", y luego se ejecuta una instrucción INPUT para recibir la respuesta a la entrada (las puestas a cero de las tres variables D, M y A que puede observar en la línea 110 son una precaución para evitar que permanezcan en estas variables eventuales valores de una fecha anterior). Resulta conveniente siempre poner a cero ("hacer un reset" de) las variables que se utilicen en operaciones de entrada para evitar que, si pulsáramos inmediatamente la tecla Return (o Enter o la equivalente), estas variables pudieran conservar un antiguo valor.

Inmediatamente después de la introducción de la fecha llamamos a la subrutina para su comprobación: GOSUB 230. El número 230 es el equivalente al "nombre de la subrutina" y es justamente el de la primera línea de la subrutina. Si tuviéramos en el programa otras subrutinas, tendríamos que definirlas con otros números de línea, como es lógico. El programa salta, pues, para ejecutar la línea 230, pero en su interior el ordenador "toma nota" del número de la siguiente línea (170) a la que retornará al final de la subrutina.

```
10 REM *************************
20 REM *  EJEMPLO DE SUBRUTINA  *
30 REM *************************
40 REM
100 PRINT "DAME LA FECHA COMO DD,MM,AA"
110 D=0:M=0:A=0:INPUT D,M,A
120 REM
130 REM ------LAMADA A LA SUBRUTINA---
140 REM -----DE CONTROL DE FECHA-----
150 REM
160 GOSUB 230
170 REM
180 IF CC=0 THEN PRINT "LA FECHA ES CORRECTA"
190 GOTO 100
200 STOP
210 REM
220 REM
230 REM SUBRUTINA DE CONTROL DE FECHA
240 REM
250 RESTORE:CC=0
260 IF M<1 OR M>12 THEN CC=1:GOTO 320
270 FOR K=1 TO M
```

```
280 READ LM
290 NEXT K
300 IF G<1 OR G>LM THEN CC=1:GOTO 340
310 IF A<0 THEN CC=1:GOTO 350
320 RETURN
330 PRINT "MES EQUIVOCADO"
340 PRINT "DIA EQUIVOCADO"
350 PRINT "AÑO EQUIVOCADO"
360 REM
370 DATA 31,28,31,30,31,30
380 DATA 31,31,30,31,30,31
```

Antes de pasar a examinar cómo se hizo la comprobación de la fecha, hagamos algunas observaciones. Las variables sobre las que actúan las subrutinas son, en principio, comunes a todo el programa, por lo que todo lo que la subrutina utiliza o calcula se pone inmediatamente a disposición del programa que efectúa la llamada, a diferencia de lo que sucede en otros lenguajes, tales como el FORTRAN, en los cuales es necesario explicitar las variables o datos compartidos y enviar y recibir los datos a y de las subrutinas. Una segunda observación es que toda subrutina se da por terminada con la instrucción RETURN. Pueden existir varias instrucciones RETURN en una subrutina, como varias instrucciones END en un programa, pero lo importante es que siempre se vuelve con ellas al programa principal y no con una instrucción de salto GOTO.

La utilización de GOTO para salir de una subrutina es un gravísimo error, aunque quizá no lo vea inmediatamente, pero que destruirá un programa cuando menos se lo espere. Ha de tenerse presente que la instrucción RETURN no tiene nada que ver con la tecla RETURN (Enter o cualquier otra equivalente), que, por el contrario, sirve para "cerrar" una línea introducida por el teclado.

Como ha podido ver, en el ejemplo usamos la variable CC para informar al programa que realizó la llamada si la fecha es correcta o no. Estas variables se denominan flags o markers (banderas, indicadores) y proporcionan información lógica binaria muy sencilla, como la de si todo ha sido correcto o hubo algo equivocado, si está bien o está mal, etc.

Dediquemos ahora algunas líneas para ver cómo comprobamos la fecha. En primer lugar, analizamos el mes (línea 260). Si hemos indicado un mes no comprendido entre 1 y 12, se pone inmediatamente CC=1 (fecha equivocada) y luego se imprime el mensaje "mes equivocado" y con la instrucción RETURN se vuelve al programa principal. Aquí, antes de proseguir, se observa el estado del indicador ("flag") CC; si es igual a cero, la fecha será correcta y el programa lo indica y prosigue (en el ejemplo, con miras a la sencilez, se pasa a solicitar otra fecha).

Volvamos a la subrutina. Si el mes es correcto, se analiza el número del día. Puesto que cada mes tiene una duración diferente se recurre al artificio de leer la duración de los meses a partir de un archivo de instrucciones DATA con un bucle FOR NEXT. Por ejemplo, para M=5, correspondiente al mes de mayo, el bucle termina con la quinta lectura, quedando en LM el valor 31. La instrucción RESTORE en la línea 250 sirve para inicializar el archivo de las instrucciones DATA antes de cualquier utilización de las subrutinas. Si el día dado a la entrada está comprendido dentro de los límites permitidos, se pasará a examinar el año. Aquí, el control es muy banal: se excluyen solamente los años negativos.

Las comprobaciones de una fecha podrían ser bastante más complejas; así se podrían tener en cuenta los años bisiestos, excluir o corregir las fechas anteriores al año 1583 (año de comienzo del calendario Gregoriano), excluir los números decimales, admitir los meses en letras, admitir solamente dos cifras para los años (tal como '28 u '85), etc.

De estas últimas consideraciones se deduce cuál es la utilidad de las subrutinas. Si quisiéramos mejorar o cambiar nuestra rutina de control sería suficiente sustituir el conjunto de sus instrucciones sin tener que perturbar todo el programa. Basta saber y recordar que las variables de entrada a la subrutina son M, D y A y que la validez de la fecha viene indicada con CC igual a cero.

Recordemos que la instrucción GOSUB se indica en los diagramas de bloques con un rectángulo con los lados verticales dobles (Figs. 3 y 4). Por el contrario, la subrutina propiamente dicha se representa con un diagrama independiente que termina con la instrucción RETURN (Fig. 4).

Niveles de subrutinas

Es preciso tratar por separado lo que concierne a la posibilidad de que una subrutina sea llamada por otra (en la jerga informática se habla de subrutinas anidadas, al igual que vimos pasaba con los bucles FOR/NEXT.

Entre las instrucciones de una subrutina puede existir una nueva instrucción GOSUB que llame a otra. Podría también, desde el punto de vista teórico, llamar a la misma subrutina de partida (el entrelazamiento resultante se denomina recursividad pero pocas veces se hace esto en la práctica.

¿Qué ocurre en estos casos? Algo parecido a lo que ocurra con el anidamiento de los bucles FOR/NEXT. Cuando el programa encuentra la instrucción RETURN de la subrutina más interior, prosigue la ejecución a partir de la instrucción siguiente a la última GOSUB leída. Después de la segunda instrucción RETURN vuelve a la GOSUB precedente, y así sucesivamente hasta volver

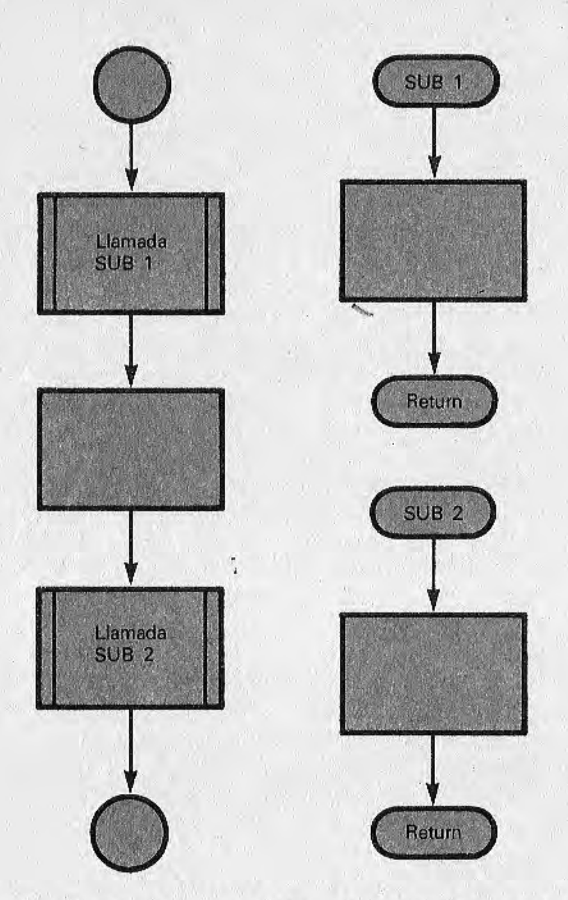


Figura 4.—Forma en que se indican las subrutinas en los diagramas de bloques.

al final al programa principal. La Figura 5 ilustra el flujo lógico en el caso de subrutinas anidadas.

Una subrutina puede ser llamada desde niveles diferentes; así, puede ser llamada por el programa principal o por otra subrutina. En cualquier caso, volverá siempre al punto desde donde partió su llamada con la instrucción GOSUB.

Existen versiones del BASIC en las que una subrutina puede llamarse a sí misma un número limitado de veces sin crear problemas; se trata de un caso de recursividad parcial. La recursividad permite aplicaciones muy interesantes, pero es un instrumento que solamente suele emplearse con los lenguajes estructurados, tal como el Pascal. En BASIC, en cualquier caso, es mejor evi-

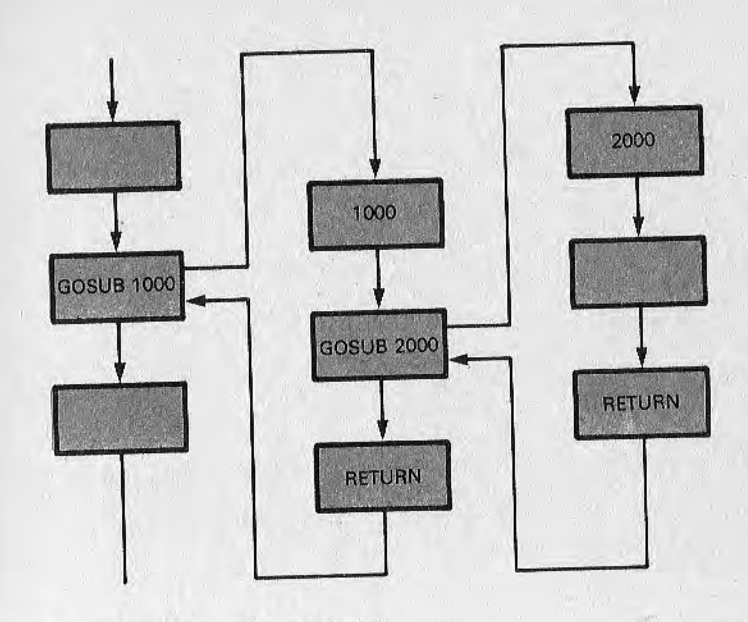


Figura 5.—Subrutina anidadas.

tar que una subrutina se llame a sí misma tanto de forma directa como a través de otra subrutina.

Por lo general, el BASIC establece un límite, bien preciso, al número máximo de niveles de subrutinas o, lo que es lo mismo, al número de llamadas una dentro de la otra. Aunque este número sea bastante elevado (16 o más), para algunas aplicaciones puede no ser suficiente por depender también del número de bucles FOR NEXT realizados.

Definición de nuevas funciones DEF FN

Después de haber hablado de subrutinas y de haber visto su utilidad para separar de un programa principal algunos grupos de instrucciones repetitivas, estudiaremos las funciones definidas por el usuario, que tienen un empleo similar al de las subrutinas.

En la primera monografía, al introducir las expresiones aritméticas indicamos que en el lenguaje BASIC están disponibles muchas funciones matemáticas que se pueden llamar con el simple empleo de su nombre. Por ejemplo, si tenemos necesidad de determinar la raíz cuadrada de un número X, bastará escribir SQR (X). En el BASIC existen funciones para el cálculo de senos, cose-

nos, tangentes, logaritmos, etc. Con un poco de atención se comprende cómo las funciones SQR, SIN, COS, TAN, LOG, etc., son subrutinas muy particulares y privilegiadas, porque cada una de ellas tiene su nombre literal, igual que una variable. Se trata de subrutinas que el fabricante del ordenador ha introducido, de una vez por todas, en el intérprete BASIC para mayor comodidad de los programadores.

Puede suceder, al escribir programas complejos y de tipo científico, que se tenga la necesidad de definir nuevas funciones que interesen solamente en un determinado programa. El BASIC le permite hacerlo con la instrucción DEF FN (definir función). La nueva función puede utilizarse posteriormente con absoluta libertad, pero solamente en el interior del programa donde se definió, lo mismo que sucedió con las subrutinas.

Veamos un ejemplo muy sencillo, pero de utilidad para este propósito. Supongamos que hay que escribir un complicado programa de geometría en el que se deben calcular, muchas veces, volúmenes de esferas. El volumen de una esfera, lo recordamos bien, viene dado por la fórmula:

$$V = (4 \times PI \times R^3)/3$$

en donde R es el radio y PI es el valor de la constante matemática π (3,141592...).

Al comienzo del programa, antes de que aparezca la necesidad de determinar el volumen de una esfera, preparamos la nueva función del modo siguiente:

```
50 DEF FN VOL(X)=(4*PI*R**3)/3
```

X es una variable ficticia ("dummy") de la definición que sirve exclusivamente para establecer sobre qué magnitud debe calcularse la función (en nuestro caso, el radio). Después de la linea 50, cada vez que se ha de calcular el volumen de una esfera se emplea la nueva función FNVOL(X), en donde el prefijo FN es obligatorio y a X se le asigna el valor del radio. Ha de tener presente que la variable argumento de la función se denomina ficticia porque actúa solamente en el interior de la función; dicho de otro modo, en el programa podría existir otra variable X que no tuviera ninguna interacción en absoluto con la otra. Así, el volumen de la Tierra, conociendo su radio de 6.380 kilómetros, viene dado por

```
100 PRINT FN VOL(6380)
RUN
1.0878 E+12
```

(1087.8 millares de millones de kilómetros cúbicos).

Evidentemente, al igual que hemos puesto un valor fijo, podríamos haber usado una variable R, por ejemplo. Así, sería:

```
100 PRINT FNVOL (R)
```

y a R le daríamos antes el valor deseado.

El nombre de una función definida en un programa debe ir precedido por FN y sigue las mismas reglas de los nombres de variables. La mayor parte de los dialectos del BASIC permiten crear solamente funciones numéricas, pero en algunos casos son posibles también las funciones de cadena. Por ejemplo:

```
50 DEF FN AST$(X$)="***"+X$+"***"
```

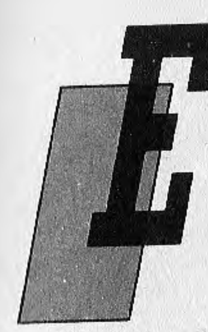
define una función de cadena que pone delante y detrás de la dada asteriscos.

La utilidad de las funciones DEF FN se aprecia en su justo valor sobre todo cuando la expresión del cálculo de la función es muy complicada, pues se evita así tenerla que volver a escribir en más puntos del programa.

CAPITULO III

OTRAS INSTRUCCIONES DE SALTO Y LOS NUMEROS ALEATORIOS

ON GOTO y ON GOSUB



Existe en el lenguaje BASIC un par de instrucciones que los puristas de la informática llaman "de salto calculado" porque el salto, o la llamada de las subrutinas, se realiza después de haber calculado una determinada variable numérica. Veamos, a título aclaratorio, un ejemplo:

```
100 ON K GOTO 150,200,220,70
110 ...
```

Cuando el programa encuentra la instrucción 100, lo primero que hace es analizar el valor de la variable K y transformarlo en un número entero. Si K es igual a 1, el programa salta a la línea 150; si es igual a 2, salta a la segunda línea indicada (200); si es igual a 3, salta a la tercera línea (220), y así sucesivamente. En nuestro ejemplo sólo se indican cuatro números de línea, y, por consiguiente, si K toma un valor superior a 4, el programa no realiza ningún salto y prosigue en la línea 110. Lo mismo sucede si K es negativo o nulo. El número de líneas que se pueden indicar después de GOTO y, por consiguiente, de saltos posibles, no tiene prácticamente ningún límite y depende exclusivamente de la longitud máxima que pueda tener una línea de BASIC en nuestro ordenador.

La instrucción ON GOSUB se comporta del mismo modo que ON GOTO, con la excepción de que llama a subrutinas; al realizarse el "retorno" prosigue con la instrucción siguiente al ON GOSUB.

La utilización de estas instrucciones es fácil de imaginar. Se pueden realizar bifurcaciones múltiples, y no solamente binarias, como ocurre con la instrucción IF THEN. Por ejemplo, para controlar un menú de opciones que ofrezca cuatro posibilidades, se puede pedir a la entrada un número del 1 al 4 correspondiente a la elección, y luego, utilizar este valor para ejecutar la parte de programa deseada. La estructura de dicho programa aparecerá como sigue:

```
100 REM **************
110 REM *    MENU    *
120 REM **************
130 REM
140 PRINT "TIENES CUATRO POSIBILIDADES"
150 PRINT "TECLEA UN NUMERO ENTRE 1 Y 4"
160 PRINT "CUALQUIER OTRO NUMERO HACE TERMINAR"
170 REM
180 INPUT X
190 REM
200 ON X GOTO 300,500,700,900
210 END
220 REM
230 REM

300 OPCION 1

500 OPCION 2

700 OPCION 3

900 OPCION 4
```

La función RANDOM (RND)

En algunas situaciones puede ser muy cómodo tener a nuestra disposición números aleatorios, no previsibles a priori (en inglés, "Random number"). Una aplicación típica la encontramos en los juegos: el ordenador extrae números al azar que el jugador no conoce y sobre los cuales está planteada la normativa del juego. Puede tratarse de una partida de "mastermind" o del vuelo de astronaves enemigas en una guerra estelar.

Para obtener números aleatorios, el lenguaje BASIC pone a disposición del usuario la función RND (abreviatura de la denominación inglesa "random"), que calcula un valor comprendido entre 0 (inclusive) y 1 (exclusive). El empleo de RND es muy fácil, puesto que basta utilizarla como una variable numérica. Cualquiera de sus llamadas, en un mismo programa, proporciona un valor aleatorio diferente.

El comportamiento de la función RND no es idéntico en todas las versiones del BASIC. Por lo general, RND(1), o poniendo como argumento en lugar del 1 cualquier otro número positivo, calcula

el valor aleatorio sucesivo, mientras que RND(0) calcula nuevamente el último valor extraído. Es decir:

```
PRINT RND(1)
  0.725372155
PRINT RND(1)
  0.384537283
PRINT RND(0)
  0.384837283
```

Es importante destacar que el generador de números aleatorios contenido en la función RND no es, en realidad, verdaderamente aleatorio (es evidente que ello estaría en contradicción con la naturaleza determinística del ordenador). Es más correcto hablar de secuencia de números pseudoaleatorios, obtenida aplicando un algoritmo particular; en la práctica, cualquier número se obtiene a partir del precedente.

Como cualquier programa, también el generador de números aleatorios (que es un programa en lenguaje máquina) requiere datos a la entrada. Precisa un número (denominado base de la secuencia) a partir del cual iniciar la cadena de números aleatorios.

A falta de esta base, el generador partirá del número contenido en una celdilla de la memoria del ordenador en el momento del encendido.

A veces es posible hacer partir la secuencia, desde un número conocido; es decir, se puede inicializar el programa generador. Esto último puede hacerse para obtener secuencias siempre iguales o siempre diferentes. Algunos ordenadores, para hacerlo, utilizan la instrucción RANDOMIZE y otros requieren un valor negativo como argumento de la función RND.

Considerando que la función RND suele generar números en el intervalo entre 0 y 1, veamos cómo es posible pasar a un intervalo diferente. Por ejemplo, para generar números aleatorios enteros entre 1 y 100 se puede escribir:

```
INT (100*RND(1)+1)
```

en efecto, 100*RND genera números entre 0 y 99,9999. Al añadir 1 se obtienen números entre 1 y 100,9999. La función INT suprime luego la parte decimal y de este modo sólo quedan los enteros entre 1 y 100.

De modo análogo se pueden obtener números aleatorios comprendidos en cualquier intervalo, incluso números negativos.

Un programa para barajar cartas

Veamos un programa que utiliza la función RND para generar números aleatorios. En este caso vamos a simular la operación aleatoria de mezclar los naipes o barajar.

Supongamos, con fines prácticos, que las 52 cartas de un mazo de baraja francesa están numeradas del 1 al 52.

Distribuir estas cartas equivale, en la práctica, a generar al azar los números comprendidos entre 1 y 52.

Damos el programa en una primera versión, más sencilla, que se limita a imprimir las cartas a medida que salen o se dan. Si se quiere conservar el mazo mezclado, basta utilizar un nuevo vector en el cual introduciríamos las cartas mezcladas:

```
100 REM *************************
110 REM *    BARAJANDO CARTAS    *
120 REM * LAS CARTAS SE INDICAN  *
130 REM * CON LOS NUMEROS 1 A 52 *
140 REM *************************
150 REM
160 DIM F(52):REM INDICADOR DE CARTAS APARECIDAS
170 REM
180 FOR I=1 TO 52
190 C=INT(52*RND(1)+1)
200 IF F(C)<>0 THEN GOTO 190
210 F(C)=1:PRINT C:REM IMPRIME LA CARTA
220 NEXT I
```

En algunas versiones del BASIC, si este programa se ejecuta varias veces, la sucesión de cartas es siempre la misma, por cuanto que la función RND se comporta siempre del mismo modo y genera la misma secuencia de números. A menudo es necesario, como en el caso de los juegos de azar, que los números aleatorios sean siempre diferentes, incluso entre sucesivas ejecuciones del programa. Para obtenerlo basta anteponer la instrucción RANDOMIZE que hace partir "al azar" los números de la función RND (por ejemplo, podríamos colocarla en la línea 175).

En un programa de esta clase, la única dificultad de programación consiste en evitar generar números iguales, pues significaría que daríamos dos veces la misma carta. Para solventar este inconveniente se utiliza una variable vectorial F() como indicador ("Flag") de las cartas que se han dado ya; el indicador F se controla en la línea 200. En la línea 190 se genera al azar un número entero comprendido entre 1 y 52 (una carta de juego) y si la carta salió ya (F<>0) se obtiene otro número. En caso contrario, el número de la carta se imprime y el indicador F(C) correspondiente se pone igual a 1.

El jugador profesional

Si ejecuta el programa que acabamos de describir, se percatará de que la velocidad con la que se dan las cartas es muy grande al principio y luego se hace menor. Ello se debe al hecho de que la función RND genera siempre un valor entre 1 y 52, por lo

que, a medida que se van dando las cartas, es preciso esperar más tiempo para encontrar los números que todavía no han salido. Este problema se resuelve con un vector M() que tiene en cuenta las cartas que han salido ya y que se va acortando de forma sucesiva.

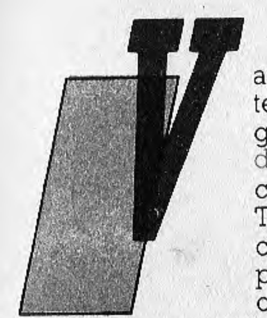
Por este motivo, en la segunda versión del programa, el bucle FOR de la línea 220 tiene el paso (STEP) negativo. La función RND, para cualquier paso, se utiliza para extraer un número dentro de un intervalo cada vez más pequeño: entre 1 y 52, entre 1 y 51, 50, 49, y así sucesivamente. El vector M() se carga inicialmente con las 52 cartas en las líneas 180, 190 y 200, y luego, a medida que salen las cartas, se llevan a M() los valores altos que van a sustituir a los que acaban de salir (línea 300).

```
100 REM *************************
110 REM *    BARAJANDO CARTAS    *
120 REM * LAS CARTAS SE INDICAN  *
130 REM * CON LOS NUMEROS 1 A 52 *
140 REM *************************
150 REM
160 DIM M(52)
170 REM
180 FOR L=1 TO 52
190 M(L)=L
200 NEXT L
210 REM
220 FOR J=52 TO 1 STEP -1
230 REM
240 REM Random genera un valor
250 REM solamente entre las cartas que
260 REM todavia no han aparecido
270 REM
280 K=INT(J*RND(1)+1)
290 PRINT M(K):REM IMPRIME LA CARTA
300 M(K)=M(J)
310 NEXT J
```

CAPITULO IV

DATOS Y FICHEROS

¿Qué son los ficheros?



Vamos a hablar en este capítulo de uno de los temas más atractivos y complejos de la programación. Los ficheros son archivos de grandes cantidades de datos que, a veces, pueden contener incluso millones de informaciones. Tanto por sus grandes dimensiones intrínsecas como para poderlos conservar durante largos períodos de tiempo, los ficheros se almacenan casi siempre en las memorias exteriores del ordenador: cinta magnética, disco rígido, casete o disco flexible. Estos dos últimos soportes son las memorias que se suelen utilizar en los ordenadores personales, tanto por su muy reducido coste como por la sencillez de su uso.

Como ejemplo más inmediato podemos citar el hecho de que también los programas son tipos particulares de ficheros. En efecto, para conservarlos, cuando apagamos el ordenador, tenemos que "salvaguardarlos" en una memoria exterior. Lo mismo puede decirse de los archivos de datos, con la salvedad de que su control, escritura y lectura se realiza con instrucciones más complejas.

Lamentablemente, la gestión de los ficheros es una de las materias que sufre más las diferencias entre las versiones del BASIC, aunque actualmente hay una cierta tendencia a uniformar las instrucciones que los controlan, gracias a la aceptación obtenida por importantes sistemas operativos, tales como MS-DOS o CP/M, o por el BASIC de Microsoft. Pero las diferencias siguen siendo bastante grandes.

Otro motivo de disparidad en el empleo de los ficheros, como

veremos mejor más adelante, se deduce del hecho de que no todos los ordenadores personales o caseros emplean, por motivos de coste, los discos flexibles, que son las memorias exteriores más cómodas y más "naturales" para la utilización de los ficheros.

No obstante estas notables diversidades prácticas, es posible hablar de ficheros en general y escribir programas que, con pocas modificaciones, son fácilmente adaptables a muchos ordenadores. Como siempre en casos similares a éste, si se comprenden los conceptos teóricos básicos, no resulta difícil "descender" a las situaciones concretas particulares que se encuentran programando cualquier ordenador personal.

Antes de hablar de la programación de los ficheros, veamos unas cuantas ideas sobre las cintas de casete y los discos flexibles, así como sobre los criterios de organización de los datos y las estructuras de los ficheros.

Cintas de casete y discos flexibles

Dijimos anteriormente que, entre los periféricos de un ordenador, los que resuelven mejor el problema de conservar los datos durante un largo período de tiempo son los que utilizan la grabación magnética. Todos nosotros sabemos cuán fiel y duradera puede ser la grabación de un concierto. El mismo principio es válido también para los ordenadores, con la única diferencia de que los datos (las señales) no se graban de modo analógico, sino digital, como es característica de la naturaleza del ordenador.

La grabación analógica es la más inmediata y fácil de intuir: un micrófono genera una señal eléctrica proporcional (analógica)

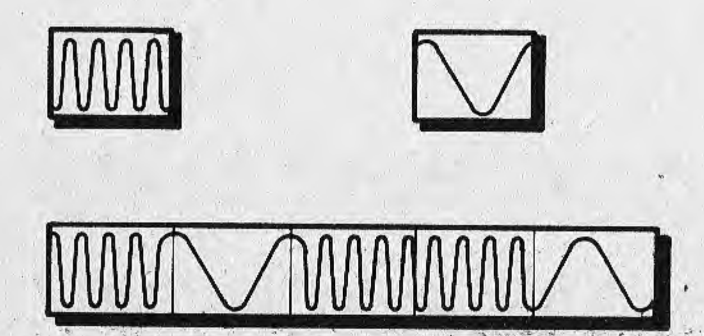


Figura 1.—Los valores 1 y 0 de los bits se almacenan como señales sinusoidales de frecuencia diferente.

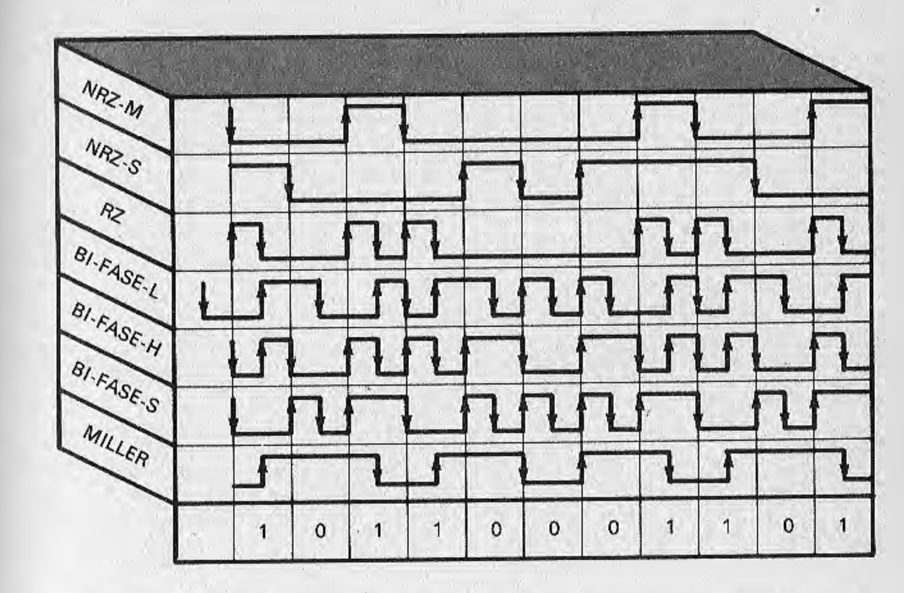


Figura 2.—Técnicas utilizadas en el almacenamiento magnético digital.

al sonido y, a su vez, la cabeza de la grabadora genera un campo magnético proporcional a la señal eléctrica. Cuando más alta sea la fidelidad con la que el micrófono convierte el sonido y cuanto más preciso sea el campo magnético generado por la cabeza, tanto más alta será la calidad de la grabación.

Por el contrario, los ordenadores trabajan con señales digitales (largas secuencias de unos y ceros), por lo que se ha tenido que transformar la tecnología magnética analógica en una tecnología digital. En las cintas, en las casetes o en los discos flexibles, los datos del ordenador son grabados como señales que representan los valores 1 o 0 de los bits (Fig. 1). El problema de la fidelidad, tal como se considera en el campo analógico, es prácticamente inexistente en los ordenadores, puesto que para que una grabación sea perfecta basta con que ningún bit 1 ó 0 se pierda.

Algunas veces, para obtener estos resultados se recurre a técnicas complejas de grabación magnética, algunas de las cuales están representadas en la Figura 2.

La grabación magnética digital puede efectuarse en dos clases de soportes: cintas y discos. A grandes rasgos, las cintas para ordenador son muy similares a las musicales. Tanto es así, que las casetes, que son las únicas cintas utilizadas por los ordenadores

personales, son muchas veces las mismas en ambos casos. A menudo, por economía, se utilizan con los ordenadores personales incluso los magnetófonos a casetes portátiles ordinarios.

La grabación magnética digital puede también realizarse, y con resultados mucho mejores, en soportes en forma de discos. En este caso la organización de los datos es mucho más compleja, aunque, en compensación, se tiene un mayor rendimiento en su manejo.

Grabación de datos en disco

Con independencia del tipo de disco (se trate de un grueso disco rígido o de un pequeño disco flexible), la organización física de los datos es prácticamente la misma siempre. Un disco está, en condiciones ideales, subdividido en muchas *pistas* circulares concéntricas formadas, a su vez, por *sectores* angulares. Los datos son grabados en bloques de longitud constante e igual a la de un sector. En la Figura 3 se ilustra la subdivisión de la superficie del disco en pistas y sectores, y en la Figura 4 se observa la correspondencia física entre un disco y su organización.

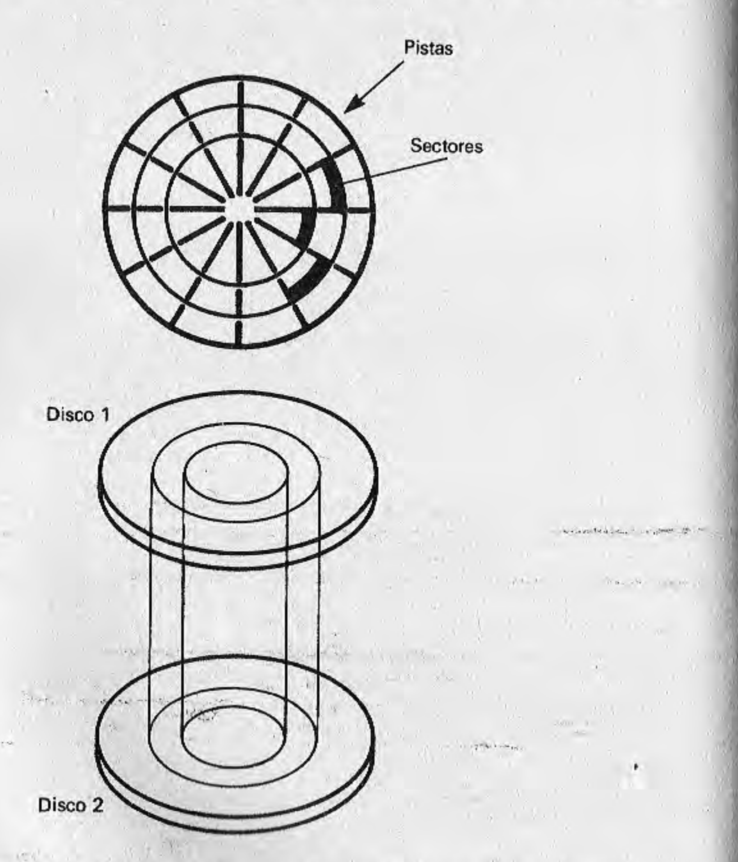
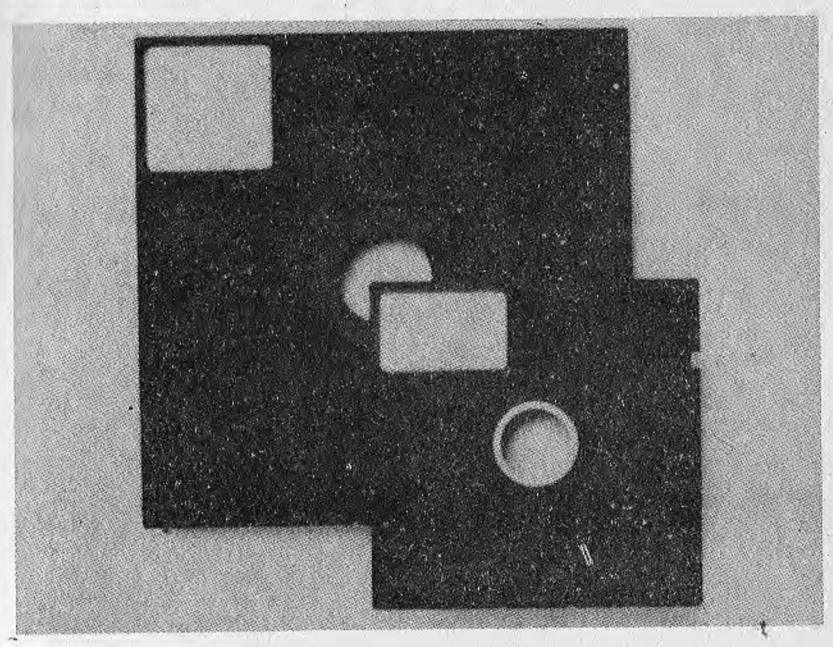


Figura 3.—Subdivisión de la cara de un disco en pistas y sectores.



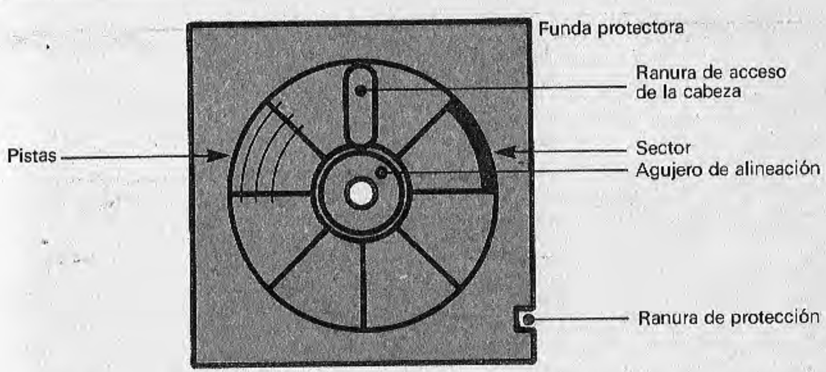


Figura 4.—Correspondencia física entre el disco y su organización.

El contenido de un sector es el bloque mínimo de datos que puede leerse o escribirse en el disco. Dicho de otro modo, no es posible tener acceso a un byte individual como se hace con la memoria central, sino que es preciso leer o escribir un sector completo que, evidentemente, está constituido por muchos bytes.

Para leer un determinado sector, se desplaza la cabeza de lectura de la unidad mecánica en la que está introducida el disco hasta el punto correspondiente a la pista deseada y luego se espera a que el sector que se busca pase por debajo de la propia cabeza, momento en el cual se lee su contenido. Lo mismo se haría para grabarlo.

Veamos la terminología utilizada, válida para toda clase de discos:

- cara ("side") es un lado o superficie del disco susceptible de grabación. Un disco puede utilizar una o dos de sus caras;
- pista ("track") es una circunferencia imaginaria en la cual se graban los datos. En una cara del disco hay muchas pistas concéntricas (recordemos que la grabación en los discos musicales se realiza, en cambio, a lo largo de una espiral);
- sector es la parte de una pista obtenida subdividiendo la circunferencia en un determinado número de sectores angulares. Un sector contiene el bloque mínimo de datos manejables durante una operación de lectura o escritura.

La capacidad total de almacenamiento de una unidad de disco depende principalmente de:

- tamaño del disco;
- número de caras utilizadas. Para cada cara se debe tener una cabeza, y, por consiguiente, aumentará el coste de la unidad de disco;
- número de pistas por cara;
- densidad de grabación de los bits a lo largo de la pista; es decir, cuántos bits se consigue grabar en un centímetro de longitud;

Tipos de discos

Aun cuando muchas unidades de disco tengan un aspecto físico diferente, todas ellas funcionan del mismo modo desde el punto de vista conceptual. No obstante, no debemos olvidarnos de hacer una puntualización que para algunos podría parecer banal: el término "disco" se utiliza para indicar tanto el soporte magnético en el que se graban los datos, como la unidad periférica

que efectúa las grabaciones o las lecturas de los datos (la denominada unidad de disco o *drive* en inglés). En cualquier caso, si no se empleara la distinción disco/unidad de disco, el significado queda claro por el contexto en el que se emplea la denominación.

Un disco, en la práctica, es un conjunto de sectores a los que se puede tener acceso de modo independiente y en un tiempo relativamente corto. Se trata, pues, de una memoria de acceso aleatorio similar, conceptualmente, a una memoria RAM, pero mucho más lenta.

En lo que respecta a la capacidad de memoria de un disco, un sector puede contener desde 128 a 1.024 bytes, mientras que un disco puede tener una capacidad total desde varias decenas o centenas de kilobytes (mil bytes) a millones de bytes, según el número de pistas utilizadas. Pasemos revista con rapidez a las clases de disco más comunes:

- Discos rígidos fijos, construidos con la tecnología Winchester. Se suelen utilizar con los grandes ordenadores, aunque actualmente, en sus versiones más pequeñas, se están difundiendo también entre los ordenadores personales por lo compactos que resultan y su notable fiabilidad.
- Diskpack. Son pilas de discos rígidos superpuestos y solidarios entre sí. La grabación suele realizarse en todas las caras, con una cabeza por cada cara. La capacidad de estos soportes es muy grande: decenas y centenares de megabytes. Suelen ser extraíbles de la unidad de disco, como lo son los discos de cartucho ("cartridge") constituidos por un solo disco.
- Discos flexibles o disquetes ("floppy disk"). A diferencia con los discos rígidos, tienen el soporte constituido por una lámina de material plástico (mylar o similar) recubierta por los habituales óxidos magnéticos y están permanentemente envueltos por una funda de plástico afelpada en su interior (ver Fig. 4). Siempre se pueden extraer de la unidad de disco. Lamentablemente, su duración no es comparable con la de los discos rígidos, tanto por estar en contacto con la cabeza de grabación, que se desplaza por encima y con el transcurso del tiempo los desgasta, como por no estar protegidos contra el polvo ambiental. Como máximo, almacenan solamente un millón de bytes (Fig. 5), pero, en compensación, su coste es verdaderamente moderado. Los primeros discos flexibles tenían un diámetro de 8 pulgadas (como un disco musical de 45 revoluciones), pero inmediatamente después se produjeron los de 5 1/4 pulgadas (denominados minifloppy), adoptados luego por los ordenadores personales. Recientemente han aparecido los "microfloppy" (de diámetros variados, por ejemplo, 3,5 pulgadas), utilizados, entre otros, por el ordenador Macintosh de Apple.

Hay que aclarar, de todas formas, que no es del todo exacto denominarlos "floppy" porque no son flexibles, sino más bien rígidos.

Como se observa en la Figura 5, la capacidad de almacenamiento de un disco, además de depender del número de caras y del número de pistas, es función también de la densidad de grabación de los bits en las pistas.

Formateado y directorio

En un disco no hay nada que indique de forma visible las pistas y los sectores donde está grabada la información. Su superficie está recubierta por igual de óxidos magnéticos con su color pardo característico. ¿Cómo es posible, entonces, escribir los datos en las posiciones correctas?

En lo que respecta a la subdivisión en pistas no hay grandes problemas, puesto que la mecánica de precisión de la unidad de disco está en condiciones de desplazar la cabeza hasta situarla sobre la pista deseada. El número de pistas distinguibles depende solamente de la precisión de la mecánica; en los discos flexibles

Ordenador y sistema operativo	Cara útil	Densidad	Pistas	Sectores	Bytes por sector	Bytes totales
Apple II DOS 3.3; PRODOS	1	Doble	35	16	256	143.360
IBM-PC MS-DOS 1.1 MS-DOS 2.0	2 2	Doble Doble	40 40	8 9	512 512	327.680 368.640
Oliveti M20 PCOS	2	Doble	35	16	256	286.720
Apple Macintosh y Lisa II	1	Doble	80	Variable	—	409.600
Sirius/Victor MS-DOS	2	Cuádruple	80	Variable	512	1.228.800
Commodore 1541 (VIC 20-C 64 Commodore DOS	1	Doble	35	Variable	256	174.648

Figura 5.—Capacidad de los discos flexibles empleados con algunos conocidos ordenadores personales.

de 5" 1/4 son posibles 35, 40 u 80 pistas, mientras que en los de 8" hay casi siempre 77 pistas. Los discos rígidos pueden tener, en cambio, centenares de pistas gracias a un dispositivo que "detecta" la intensidad del campo magnético de la pista localizada a partir de la cabeza.

Una vez que tenemos entonces la cabeza situada sobre la pista deseada, es preciso establecer el punto de comienzo de los sectores. Esto puede conseguirse de dos maneras: señalando de forma física en el disco con orificios los sectores (la operación denominada "sectorización" por hardware) o bien con la grabación de señales especiales (la denominada "sectorización" por software). El primer método es muy utilizado con los discos rígidos ("hard disk"), mientras que el segundo se emplea en casi todos los ordenadores personales para los discos flexibles.

Con la sectorización por software los sectores se identifican mediante informaciones (en la llamada "cabecera"), grabadas en la propia pista. Estos indicadores, que no están presentes en un disco nuevo (virgen) dependen del tipo de ordenador utilizado o, más exactamente, de su software para disco, de su sistema operativo de disco (o DOS).

Así, cuando se quiere emplear un nuevo disco es preciso formatearlo (o inicializarlo) en primer lugar. Esta operación, ejecutable también en discos antiguos que se quieran reutilizar como nuevos, además de señalizar los sectores suele grabar en el disco la tabla de control de los ficheros, denominada directorio.

El directorio (o tabla índice) contiene las informaciones necesarias para identificar todos los ficheros grabados en el disco. Una parte de esta información es posible leerla con el comando DIR del BASIC (o CATALOG, según de qué ordenador se trate). La principal información contenida en un directorio es:

- nombres de los ficheros,
- tipos de fichero,
- sectores en los que se encuentran los ficheros,
- dimensiones de los ficheros,
- fecha de su creación,
- fecha de la última modificación,
- permiso de acceso a los ficheros.

La información más importante es la relativa a la identificación de los sectores en donde se encuentra el fichero; gracias a ello resulta posible tener acceso directo a los ficheros individuales o a sus registros.

Ficheros secuenciales

Casi todas las versiones del BASIC permiten emplear dos tipos de ficheros: secuenciales y directos (o aleatorios). En los ficheros secuenciales, que son los más sencillos, los datos están grabados uno tras otro, como si el dispositivo de memoria de masa fuera siempre una cinta magnética (Fig. 6). La información contenida en el fichero puede, a su vez, subdividirse en registros y campos, pero, en cualquier caso, se grabarán o volverán a leerse siempre en orden a partir del comienzo del fichero. Funcionan como si se tratara de una casa de vecinos: una vez entramos en el portal (fichero), si queremos llegar al tercer piso (registro) debemos pasar antes por el primero y el segundo (Fig. 7).

Con frecuencia, un fichero secuencial se crea "imprimiendo" los datos en el propio fichero con la instrucción PRINT# equivalente a la instrucción PRINT normal, que visualiza, en cambio, los datos en la pantalla. Un fichero secuencial se asemeja, a todos los efectos, a una larguísima cadena de caracteres.

Para leerlo se debe partir siempre desde el comienzo utilizando, de forma secuencial, instrucciones INPUT# completamente similares a las instrucciones INPUT. En la práctica, los datos se leen desde el fichero como si se leyeran a partir del teclado.

Un dato puede ser también un registro completo, constituido por varios campos. En tal caso, es preciso para cada registro escribir (o leer) uno tras otro, en su orden, todos los campos que lo componen.

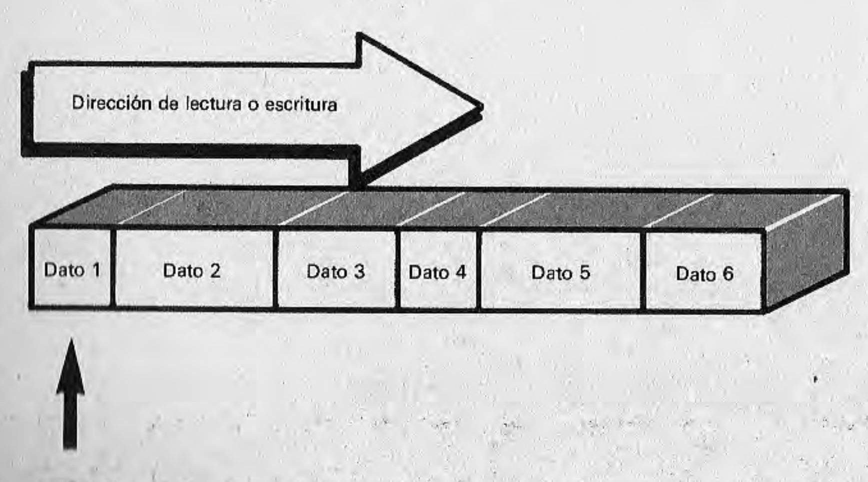


Figura 6.—En un fichero secuencial, para tener acceso a un registro concreto se debe pasar a través de todos los precedentes.

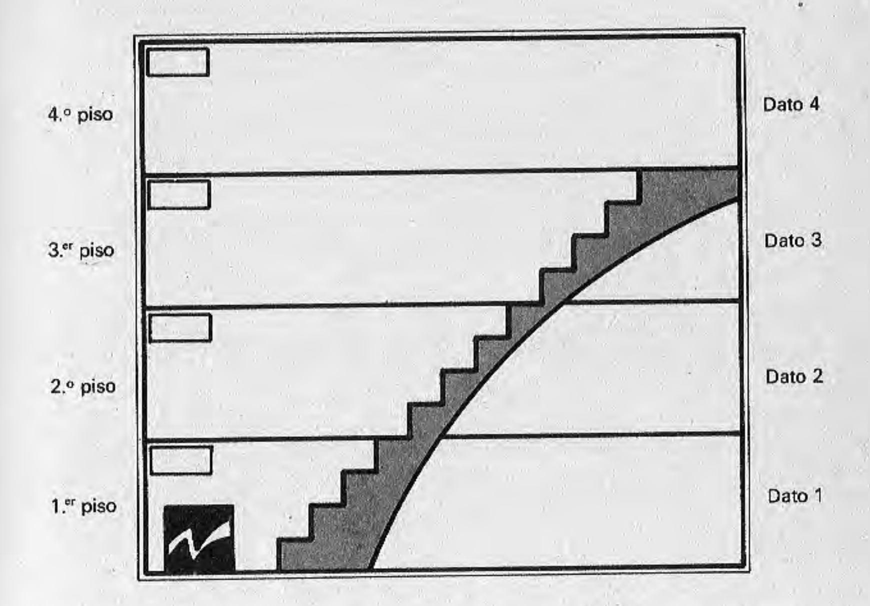


Figura 7.—Al igual que cuando en una casa queremos llegar al tercer piso tenemos que pasar por el 1.º y el 2.º, en un archivo secuencial para alcanzar un dato debemos antes recorrer todos los anteriores.

Los ficheros secuenciales sirven para todas las aplicaciones en las que no es fundamental tener acceso directo a los datos situados en puntos intermedios y se puede recorrer el fichero desde el comienzo al final. Un caso particular de fichero secuencial, al que ya hicimos alusión, es el de los programas. Cuando se da la orden SAVE, el programa se almacena lo mismo que un fichero secuencial, al menos desde el punto de vista conceptual, aunque el BASIC utiliza órdenes especiales para realizar esta operación (comandos SAVE y LOAD).

Ficheros directos

El segundo tipo de fichero es el de los ficheros *directos* o *aleatorios*. Estos ficheros tienen la gran ventaja de que permiten el acceso directo a todos sus datos del mismo modo y con el mismo retraso. Por este motivo, en inglés se les denomina "random" (aleatorios), precisamente por su posibilidad de alcanzar inmediatamente un punto cualquiera del fichero elegido al azar, sin ninguna limitación. Es como cuando (Fig. 8) nos acercamos a nuestra có-

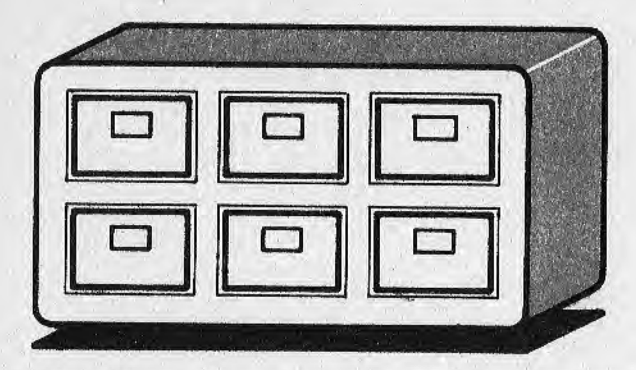


Figura 8.—La forma de manejar un registro directo es parecida a la que usamos cuando buscamos algo en nuestra cómoda: podemos abrir el cajón que mejor nos parezca sin seguir ningún orden.

moda; podemos abrir con la misma facilidad cualquiera de sus cajones (si ninguno está atascado, claro).

Con respecto a los ficheros secuenciales, los directos son más prácticos y eficaces, pero tienen también una estructura más compleja y rígida; en los ficheros directos la información debe estar forzosamente organizada en bloques de longitud fija (los denominados registros). Dicho de otro modo, los ficheros directos están constituidos por una serie de registros, cada uno de los cuales puede contener una cantidad bien definida de información, que se establece de una vez por todas cuando se crea el fichero (Fig. 9).

Campo 1	Campo 2	Campo 3		Campo 4	Registro 6
Apellido	Nombre	Domicilio		Año nacimiento	
Campo 1	Campo 2	Campo 3		Campo 4	Registro 7
Apellido	Nombre	Domicilio		Año nacimiento	
Campo 1	Campo 2	Campo 3		Campo 4	Registro 8
Apellido	Nombre	Domicilio		Año nacimiento	
Campo 1	Campo 2	Campo 3		Campo 4	Registro
Apellido	Nombre	Subcampo Calle	Subcampo Ciudad	Año nacimiento	

Figura 9.—Los ficheros directos, o aleatorios, están constituidos por registros que tienen todos ellos la misma longitud.

Por ejemplo, supongamos que los registros de un fichero directo tengan que contener solamente nombres y apellidos. Podemos tener personas con el nombre muy corto ("Juan Paz") y otros con el nombre muy largo ("José Antonio Campuzano"). El primero está constituido por ocho caracteres (incluyendo el espacio), mientras que el segundo tiene veintidós. Si queremos que el fichero contenga estos nombres, todos los registros deberán tener una longitud fija (mínima) de veintidós caracteres (o veintitrés para algunos ordenadores que añaden un carácter separador entre los registros). Esto quiere decir que para el primer nombre desperdiciamos 14 bytes, lo cual es un despilfarro considerable. No hay ninguna otra alternativa: en los ficheros directos: o cortamos los nombres más largos o desperdiciamos espacio con los más cortos.

Estructura y acceso

Hasta ahora hemos utilizado los términos secuencial y directo para referirnos tanto a la estructura del fichero como al método de acceso a los datos. En realidad, se trata de dos conceptos bien distintos y habría sido más exacto hablar de:

- estructura del fichero (secuencial o directa). Forma en que están organizados, es decir, escritos, los datos;
- método de acceso (secuencial o directo). Forma de acceder a los datos del fichero, tanto en la fase de escritura como en la de lectura.

Es fácil intuir, y también recordar cuando se escriben los programas, que en un fichero con estructura secuencial sólo se puede tener un acceso secuencial, mientras que en un fichero con estructura directa se puede acceder de cualquier modo.

Por consiguiente, la elección de una estructura o de otra depende no solamente del tipo de datos que se tengan, sino también de la clase de empleo que les vayamos a dar y del tipo de ordenador que utilizamos. Por ejemplo, los más pequeños ordenadores caseros no suelen permitir controlar ficheros directos y, a veces, ni siquiera secuenciales.

En este punto, volvemos a hablar de ordenador personal. Estos ordenadores suelen trabajar sólo con magnetófonos a casete (como las musicales) y, por consiguiente, no permiten controles sofisticados (¡es un decir!) de los ficheros. Para hablar de ficheros, como se habrá percatado ya, es preciso, en la práctica, referirnos a los ordenadores que utilizan discos flexibles.

Antes de proseguir para ver finalmente las instrucciones BASIC relativas a los ficheros, hagamos una puntualización técnica con respecto al empleo de los magnetófonos a casete ordinarios (musicales, para entendernos). Para controlar los ficheros secuenciales, no todos estos magnetófonos proporcinan resultados satisfactorios. De hecho, un fichero secuencial es una sucesión de registros independientes entre sí, que deben escribirse y leerse de manera secuencial, pero no necesariamente todos con continuidad temporal: entre la escritura, o lectura, de un registro y el sucesivo puede existir una larga pausa; basta considerar el tiempo necesario para preparar un nuevo registro antes de escribirlo.

Por este motivo, el motor de la grabadora debe poder ser *controlable* por el ordenador, que lo hará avanzar o parar cuando sea necesario. Los ficheros secuenciales en casete se pueden utilizar, pues, solamente con los ordenadores que tengan el control remoto de la grabadora. Con mayor razón, los ficheros directos nunca se pueden escribir en ningún tipo de cinta magnética, ni siquiera en las empleadas por los grandes ordenadores, porque estos ficheros requieren que se pueda tener acceso directo a cada uno de sus registros.

CAPITULO V

GESTION DE FICHEROS EN BASIC

Apertura y cierre de ficheros

Ahora que tenemos más claras las ideas sobre la estructura de los ficheros y los soportes sobre los cuales crearlos (cintas o discos) volvemos al lenguaje BASIC. Vamos a describir la forma en que hemos de proceder para controlar un fichero.

Supongamos que hay muchos datos que almacenar: por ejemplo, una lista de direcciones. Entonces, para utilizar un fichero es preciso, en primer lugar, abrirlo; es decir, avisar al ordenador de que deseamos una conexión "lógica" con un magnetófono de casete o un disco.

Una analogía válida de la apertura de un fichero es la de telefonear: cuando marcamos un número y la persona llamada descuelga el teléfono, se establece una conexión y queda "abierta" la comunicación.

Solamente después de que un fichero se ha abierto es posible realizar en el mismo las operaciones de lectura o escritura de los datos. En ese preciso momento se hace accesible ("transparente") para el usuario y, en un cierto sentido, puede compararse con una gran expansión de la memoria central. Al acabar las operaciones de lectura y de escritura el fichero deberá cerrarse para cortar la conexión lógica que se había establecido con la apertura inicial.

Digamos inmediatamente algo muy importante: un fichero, una vez abierto, lleno de datos y cerrado se conservará aún cuando termine el programa que lo ha escrito. Los ficheros tienen "vida propia" gracias al hecho de que están depositados en memorias

magnéticas y, por consiguiente, no pierden la información al apagar el ordenador. Una cinta de casete (o un disco flexible) conservan datos y programas durante muchos años, lo mismo que sucede en el caso de las grabaciones musicales. Es muy raro que un soporte magnético pierda la información si se tomaron algunas precauciones mínimas, como no mojarlo, calentarlo o ensuciarlo.

Un fichero, además de ser un gran medio de almacenamiento de nuestros datos, puede hacer también de puente de conexión entre dos programas. Por ejemplo, si dos programas son demasiado amplios para residir ambos en la memoria central (como un programa único) pueden cargarse, uno a uno, en el ordenador e intercambiar sus datos a través de un fichero temporal (Fig. 1).

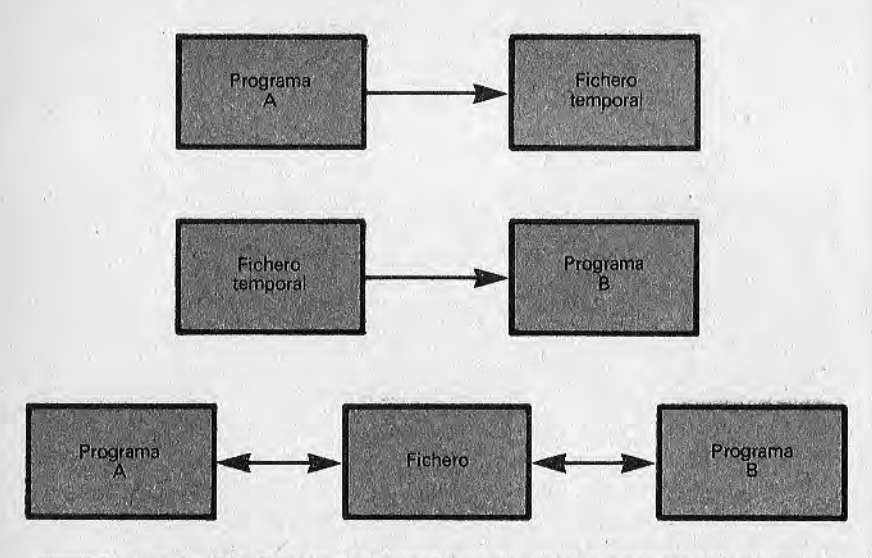


Figura 1.—Un fichero temporal puede usarse para transferir los datos desde un programa a otro.

Cuando se trabaja con discos flexibles se pueden utilizar (abrir) simultáneamente varios ficheros y desarrollar varias operaciones de lectura y de escritura de datos. Es evidente que cada fichero abierto deberá tener un nombre diferente (un número de teléfono diferente en la analogía de la llamada telefónica). El número máximo de ficheros simultáneamente abiertos no es el mismo para todos los ordenadores.

OPEN

La primera instrucción del BASIC que encontramos para controlar los ficheros es OPEN. Esta instrucción OPEN es la que avisa al ordenador de que queremos trabajar con un fichero.

Las reglas que rigen su empleo varían mucho según la versión del BASIC, pero todas ellas tienen en común que deben establecer las características del fichero y en qué periférico (casete o disco) debe efectuarse la grabación o lectura de los datos. Los parámetros de las instrucciones OPEN son:

- tipo de fichero (secuencial o directo);
- tipo de operación (lectura o escritura de los datos);
- un código numérico denominado canal o fichero lógico, que se asociará a dicho fichero para las sucesivas instrucciones de lectura o escritura de los datos (PRINT#, INPUT#, etc.). Si se utilizan varios ficheros simultáneamente este código sirve también para distinguirlos;
- nombre del fichero, y
- periférico con el que se debe realizar la cenexión.

Es bastante habitual que en los ordenadores que admiten ficheros secuenciales y directos el formato de las instrucciones OPEN no coincida.

Veamos algunos ejemplos, con la indicación de a qué ordenador se refieren.

Commodore 64:

OPEN <nº canal>, 1, <modo>, "<nombre>".

```
100 OPEN 5,1,0,"AGENDA"
```

Quiere decir que el fichero con el nombre AGENDA se escribe (lo que se indica por 0 = Output, archivo de salida) desde la cinta del casete, única opción posible (lo que se indica por 1) a través del canal lógico 5.

```
100 OPEN 4,1,1,"TELEFONO"
```

En este caso, el fichero TELEFONOS se lee (I = input, de entrada) en la cinta de casete (1) a través del canal lógico 4.

Spectrum:

Este ordenador utiliza la instrucción OPEN solamente con las unidades Microdrive (pequeñas casetes que trabajan en bloques como los discos).

En las unidades de casete no es posible abrir ficheros, aunque se pueden grabar en ellas bloques completos de datos bajo la forma de vectores o matrices con la instrucción SAVE DATA. Por ejemplo:

```
100 SAVE "DIRECCIONES" DATA A$()
```

Hace que la matriz de cadena A$ () se almacena en cinta de casete con el nombre de DIRECCIONES. Ha de tener en cuenta que la palabra DATA significa "datos" y no debe confundirse con la instrucción DATA READ.

Para la lectura de la matriz completa A$ se emplea la instrucción LOAD DATA:

```
100 LOAD "DIRECCIONES" DATA A$()
```

Apple II:

Los ordenadores de Apple hacen uso prevalente de discos flexibles. La instrucción de apertura de un fichero con el DOS 3.3 (el software operativo para discos del Apple II) debe darse a través de una instrucción PRINT, haciendo preceder el comando OPEN por un carácter Control D (CTRL-D). Para un fichero secuencial sería:

```
100 D$=CHR$(4):REM <CTRL D>
110 PRINT D$;"OPEN DIRECCIONES"
```

Con lo anterior se abrirá en el disco el fichero secuencial DIRECCIONES. El DOS 3.3. no utiliza números de canal. Disculpe la aparente dificultad de estas instrucciones, pero a menudo ¡hay que tomar las cosas tal como son! Para no adentrarnos demasiado en el funcionamiento de las máquinas individuales, no hemos dado la explicación detallada de todos los parámetros de las instrucciones OPEN: ¡de vez en cuando es necesario consultar los manuales de los ordenadores!

La apertura de los ficheros directos se consigue en el Apple añadiendo el parámetro L, que indica la longitud del registro:

```
100 D$=CHR$(4):REM <CTRL D>
110 PRINT D$;"OPEN DIRECTO,L50"
```

Esto quiere decir que el fichero DIRECTO, de estructura aleatoria, tiene los registros con una longitud de 50 caracteres cada uno.

IBM PC:

Este ordenador trabaja con el BASIC Microsoft, que suele utilizarse con el sistema operativo MS-DOS. Las instrucciones relativas a los ficheros son muy sencillas y elegantes. Para una secuencial:

OPEN "<modo>", # <canal>, "<nombre>"

```
100 OPEN "O",#5,"B:ESTUDIANTES"
```

El fichero secuencial ESTUDIANTES se abre con el número de canal 5 en la unidad B. Las operaciones son de escritura (la "O" indica salida del ordenador).

```
100 OPEN "I",#5,"B:ESTUDIANTES"
```

Como la anterior, pero las operaciones son de lectura ("I" por imput, igual entrada).

Para los directos:

OPEN "R", # <canal>, " <nombre>", <longitud>

```
100 OPEN "R",#4,"A:VUELOS.1",230
```

El fichero directo VUELOS.1 se abre con el número de canal 4. El parámetro "R" indica que el fichero es directo. Los registros tienen una longitud de 230 caracteres. Las operaciones pueden ser de lectura o de escritura.

Equipos MSX

Para los secuenciales (desde casete) usan:

OPEN "<nombre>" FOR <modo> AS <canal>

y para los directos:

OPEN "<nombres>" [FOR <modo>] AS [#] <canal> [LEN=<longitud>]

CLOSE

Al final de las operaciones de lectura o escritura de cualquier tipo de fichero, es imprescindible *cerrar el canal* o, dicho de otro modo, desconectar lógicamente el magnetófono a casete o el disco flexible (...equivale a colgar el teléfono).

La instrucción del BASIC es, en este caso, muy sencilla: CLOSE seguida por la indicación del número de canal incluido en la intrucción OPEN correspondiente (atención a esta correspondencia). Por ejemplo:

```
500 CLOSE #1
```

Sirve para cerrar el canal 1.

Solamente el ordenador Apple con DOS 3.3. presenta una complicación, debida a la obligación de dar el carácter Control D con PRINT:

```
500 PRINT CHR$(4);"CLOSE"
```

Cuando un fichero está abierto como de lectura, la falta de cierre, CLOSE, no produce ningún daño efectivo. Por el contrario, en el caso de escritura, la falta de cierre puede dar lugar a la pérdida del último bloque de datos, que no se graba en el fichero.

El BASIC y los ficheros secuenciales

Después de haber visto las instrucciones de apertura y de cierre, tanto para los ficheros secuenciales como para los directos, pasamos a las instrucciones necesarias para escribir y leer los registros individuales; primero veremos las correspondientes a los ficheros secuenciales y después las relativas a los ficheros directos.

PRINT#

Escribir los datos en un fichero secuencial del BASIC es muy sencillo. Se procede, desde el punto de vista conceptual, como si los datos se visualizaran en la pantalla: cada línea equivale a un registro.

La instrucción PRINT# se comporta como PRINT, pero en lugar de enviar los datos a la pantalla los envía al fichero (atención al signo # denominado "cancela", que distingue a las dos instrucciones).

Para obtener la separación de los registros en campos, reconocibles luego en la fase de lectura por una instrucción INPUT#, es preciso escribir entre los valores de cada campo, de forma expresa, comas, puntos y coma u otros separadores válidos según el ordenador de que se trate. Veamos un ejemplo en el que se "imprimen" en el fichero, abierto antes con el número de canal número 6 (#6), dos campos C1$ y C2$:

```
200 PRINT #6,C1$;",";C2$
```

La grabación de la coma "," (separador de campo) es fundamental para permitir al ordenador, cuando las lea después, separar las dos cadenas C1$ y C2$. Sin esta coma, el ordenador consideraría que se trata de una cadena única, unión de C1$ y C2$. Cuando se graban cadenas es preciso prestar atención a su longitud máxima, que varía de un ordenador a otro. Estas limitaciones dependen, en gran medida, del tipo de instrucciones que se utilizarán luego para leer el fichero. Por ejemplo, con la instrucción INPUT de Commodore no se pueden leer más de 80 caracteres, incluido el carácter Return, situado al final de un registro como separador.

Ejemplo de escritura en un fichero secuencial

Le proponemos un ejemplo "sui generis" de fichero secuencial: la escritura de un fichero en el que están incluidos los elementos de una matriz rectangular de números. La grabación se iniciará desde el principio, línea por línea.

Sabemos que en BASIC las matrices son tablas (o "arrays") que residen en la memoria central, como todas las variables; pero pueden existir dos motivos válidos para querer que los datos de estas tablas se incluyan en un fichero exterior. El primero, evidentemente, es la conservación de las propias matrices, y el segundo es que las matrices muy grandes no pueden "residir" por completo en la memoria central. Por ejemplo, una matriz de 100×100 números enteros ocupa 20.000 bytes (10.000 números, cada uno de los cuales está almacenado en 2 bytes). Muchos pequeños ordenadores no pueden disponer fácilmente de tanta capacidad de memoria para una sola matriz, por lo que se tiene que probar a grabarlas en periféricos externos.

Veamos, pues, cómo escribir en un fichero secuencial una matriz, elemento por elemento y línea por línea.

Supongamos que la matriz tenga 100 filas y 50 columnas, lo que indicamos en la línea 160 con las dos variables TR y TC. En la línea 190 abrimos el fichero (las instrucciones son las del BASIC Microsoft), en donde "O" indica que el fichero está abierto en escritura ("output"), el número de canal que elegimos es el número 1 (#1) y el nombre que damos al fichero es de MAT. SEC.

Los dos bucles FOR NEXT permiten tomar en la entrada, desde el teclado, y luego escribir en el fichero uno por uno los ele-

mentos de la matriz. En la línea 270 se presenta en la pantalla el mensaje de solicitud de un elemento. Por ejemplo, para el primer elemento aparecería:

FILA 1 - COL 1:

a lo que se debe responder con el valor numérico del primer elemento. Tenga presente que estos números son de tipo real, y no entero, como dijimos anteriormente, por lo que la matriz ocupa muchos más bytes (4 bytes por cada número real en el BASIC Microsoft).

En la línea 290 el número E se graba en el fichero. Cada registro del fichero contiene, pues, un único número real.

La instrucción CLOSE#1 en la línea 330 cierra el fichero. Su existencia es fundamental, porque garantiza que el último bloque de datos se transfiera efectivamente a la memoria exterior.

Tratemos de explicar mejor este punto, al que se hizo alusión con anterioridad. La grabación de los datos en un disco (o en una cinta de casete) no tiene lugar realmente cada vez que el ordenador encuentra una instrucción PRINT#, sino solamente cuando el buffer preparado para dicho fichero está lleno de datos. No tiene por qué preocuparse, ya que estos problemas no afectan al programador, sino que son competencia exclusiva del sistema operativo DOS.

Si, por ejemplo, el buffer tiene una longitud de 256 bytes y se graban números de 4 bytes a los que se añade un carácter separador de registro (como Return, ASCII 13), dando un total de 5 bytes por regsitro (en los ficheros directos la cuestión es diferente al faltar los separadores), en el buffer cabrán 256/5=51 registros completos. El byte que sobra es inutilizado. En este caso solamente después de 51 instrucciones PRINT#1, el buffer se "trasvasará" al exterior y se verá girar el disco o la cinta de casete. Cuando se ejecute la última instrucción PRINT# es poco probable que coincida con el llenado del buffer, por lo que si en este momento apagáramos el ordenador o finalizara el programa, los últimos registros que se encuentran cargados en el buffer se perderían. CLOSE tiene precisamente el cometido de almacenar el último resto de fichero, es decir, trasvasar el último buffer, además de avisar al sistema DOS de que las operaciones en ese fichero están terminadas.

```
100 REM ***************************
110 REM *    ESCRITURA DE UNA      *
120 REM *    MATRIZ DE NUMEROS     *
130 REM * EN UN FICHERO SECUENCIAL *
140 REM ***************************
150 REM
160 TR=100:TC=50:REM MATRIZ DE 100 X 50
```



```
170 REM
180 REM
190 OPEN "O",#1,"MAT.SEC"
200 REM
210 REM ESCRITURA DE LA MATRIZ
220 REM
230 PRINT "DAR LOS VALORES DE LOS ELEMENTOS:"
240 REM
250 FOR J=1 TO TR
260 FOR K=1 TO TC
270 PRINT "FILA ";J;" - COL ";K;" :"
280 INPUT E
290 PRINT #1,E
300 NEXT K
310 NEXT J
320 REM
330 CLOSE #1
340 REM
350 END
```

Más adelante, después de que estudiemos las instrucciones de lectura de los ficheros secuenciales, veremos cómo leer la matriz.

A modo de inciso, recordemos que el ordenador Spectrum realiza la grabación de una matriz directamente con la instrucción SAVE DATA.

INPUT# y LINE INPUT#

La lectura de los ficheros secuenciales puede realizarse de dos maneras: leyendo un registro completo (INPUT# y LINE INPUT#) o bien leyendo un carácter cada vez (GET# o INPUT$). Ha de prestarse atención a esta instrucción GET#, que no debe confundirse con la instrucción GET# del BASIC Microsoft, por ejemplo, que actúa, por el contrario, sobre los ficheros directos y de la que hablaremos a continuación.

En el primer caso, INPUT# se comporta como su "prima hermana" INPUT. INPUT recibe los datos desde el teclado, mientras que INPUT# los recibe desde el fichero. Las variables, indicadas en la instrucción, deben corresponder al tipo de datos grabados en el fichero.

INPUT# <canal>, <variable1> [,<var2>...]

Las comas grabadas con PRINT# separan los campos y equivalen a las introducidas por el teclado para separar los datos a la entrada. El carácter Return que separa los registros equivale a la pulsación de Return en el teclado. Por ejemplo, para leer cuanto se escribió con la instrucción PRINT# indicada anteriormente se debe utilizar:

```
600 INPUT#2,A$,B$
```

Con LINE INPUT# se lee, en cambio, un registro completo, hasta el carácter Return. Esta instrucción se utiliza solamente con variables de cadena y cada eventual distribución lógica en campos debe hacerse por separado. La importancia de LINE INPUT# está en el hecho de que admite una línea completa con cualesquiera caracteres (comas y otros) y la asigna a una sola variable. Por ejemplo:

```
600 LINE INPUT#1,L$
```

GET# e INPUT$

Acabamos de hacer alusión al hecho de que, a menudo, es preferible leer un fichero secuencial que contenga datos del tipo de cadena, en forma de un carácter cada vez. Esto es así para evitar los inconvenientes de la instrucción INPUT#, que exige una correspondencia rígida entre sus variables y los campos del registro separados por las comas. Esta lectura carácter por carácter se obtiene por la instrucción INPUT$ o con GET#. En este caso ya no existen campos y registros del fichero, sino que la destribución lógica debe ser reconstruida por el programa de lectura.

Así, en el ejemplo siguiente, en la cadean R$ "acumulamos" un registro leyendo un carácter cada vez. Al comienzo de cada campo, la cadena R$ se pondrá a cero (línea 150); a través de la instrucción INPUT$ se lee y asigna a la variable A$, un carácter desde el canal número 1 (#1) abierto con anterioridad. El primer número 1, entre los paréntesis de la instrucción, indica precisamente que se lee un solo carácter; en realidad, INPUT$ permitiría que se leyese más de uno. Inmediatamente después se efectúan dos comprobaciones: si el carácter leído es una coma (ASCII 44), el programa pasará al control del campo, y si el carácter es Return (ASCII 13), se terminará el registro. En todos los demás casos, el carácter se añade a la cadena R$ (línea 190). Después del control de campo o de registro, el programa volverá a la línea 150.

```
150 R$=" "
160 A$=INPUT$(1,#1)
165 REM
170 IF A$=CHR$(44) THEN GOTO 200
180 IF A$=CHR$(13) THEN GOTO 300
190 R$=R$+A$:GOTO 160
195 REM
200 ... R$ Contiene un campo
    ...
300 ... Final de registro,R$ contiene el ultimo campo
```


Si se emplea la instrucción GET#, la línea 160 debe cambiarse como sigue:

```
160 GET#1,A$
```

Si el fichero contiene datos numéricos, grabados con variables numéricas, se debe utilizar INPUT# por cuanto que GET# e INPUT$, al leer un carácter cada vez, harían muy difícil la reconstrucción de los números expresados en forma exponencial (por ejemplo: 3.456 E-8).

Se podría pensar que la lectura de un fichero con GET# (un carácter cada vez) es mucho más lenta en relación con la efectuada con INPUT# (un registro completo), pero, en realidad, el tiempo empleado es prácticamente igual. GET# ofrece, no obstante, la ventaja de que cualquier cosa que se escriba en el fichero puede ser leída como si se programase en lenguaje máquina. Exige, pues, un poco más de dominio y de atención.

Ejemplo de lectura en un fichero secuencial

Pasemos ahora a leer el fichero secuencial escrito anteriormente para "magnetizar" una matriz numérica.

Incluso en este caso la primera instrucción operativa del programa debe ser la de apertura del fichero. En la línea 190 hemos utilizado OPEN con el parámetro "I" (INPUT = de entrada) seguido por el número del canal (1).

Este número que usamos ahora no depende, en absoluto, del número de canal empleado en el programa de escritura, y solamente por casualidad se eligió también el 1. En cambio, el nombre del fichero MAT.SEC debe ser exactamente el utilizado en la escritura, porque solamente así el sistema DOS puede buscar en el disco el fichero correcto que contiene la matriz deseada.

La lectura de la matriz es muy simple y basta utilizar una instrucción INPUT# con una variable numérica (variable que puede ser diferente de la empleada en la escritura). Los dos bucles FOR NEXT anidados tienen como objetivo leer los elementos de la matriz línea por línea. En la línea 310, CLOSE cierra la comunicación con el fichero. En este caso, su eventual omisión no produciría daños en los datos, como hubiera sucedido, por el contrario, en el caso de escritura.

```
100 REM **************************
110 REM *    MATRIZ DE NUMEROS   *
120 REM *  EN FICHERO SECUENCIAL *
130 REM *        LECTURA         *
140 REM **************************
```

```
150 REM
160 TR=100:TC=50:REM MATRIZ DE 100 X 50
170 REM
180 REM
190 OPEN "I",#1,"MAT.SEC"
200 REM
210 REM ---- IMPRESION DE LA MATRIZ ----
220 REM
230 FOR L=1 TO TR
240 FOR M=1 RO TC
250 INPUT #1,C
260 PRINT "FILA ";L;" -COL ";M;
270 PRINT " : ";C
280 NEXT M
290 NEXT L
300 REM
310 CLOSE #1
320 REM
330 END
```

Si en lugar de una matriz numérica se hubiera tratado de una matriz de cadenas, no habría diferencias importantes. En lugar de la variable C habríamos empleado una variable de cadena tal como E$. Las únicas complicaciones se habrían dado si en lugar de grabar un solo número por registro hubiésemos querido grabar más. En este caso hubiéramos tenido que forzar la grabación de "comas separadoras", a las que antes hicimos alusión, de este modo:

```
290 PRINT #1,A;",";B;",";C
```

en el programa de escritura.

Ello hubiera permitido, en la lectura, emplear variables separadas entre sí:

```
250 INPUT #1,D,E,F
```

Las comas utilizadas en escritura se comportan exactamente igual que las comas que se han de teclear para separar la entrada de varios datos cuando se utiliza la instrucción INPUT.

El BASIC y los ficheros directos

Hablemos ahora de ficheros directos, que solamente pueden grabarse en discos, como explicamos anteriormente. Los ficheros directos, además de tener la gran ventaja de permitir acceder directamente a sus datos, permiten también reescribir o leer, sin modificar el OPEN, cada registsro de forma individual.

Con los ficheros secuenciales (repetir estos conceptos nunca viene mal) se pueden hacer, en cambio, solamente operaciones de lectura o de escritura, de forma separada. Por estos motivos, pero también por su más compacta grabación en los discos, los ficheros directos son casi siempre preferibles.

Hagamos primero una puntualización. Hemos utilizado el término ficheros directos o de acceso directo, pero hay quien distingue más precisamente entre ficheros directos y ficheros relativos. La diferencia es solamente de tipo práctico y no de naturaleza conceptual, por lo que, después de haberla aclarado, seguiremos utilizando la primera denominación. Un fichero se denomina directo por cuanto que al permitir apuntar directamente a cada uno de sus registros está en contraposición con los ficheros secuenciales. Los americanos utilizan el término "random" (aleatorio) pretendiendo indicar que el tiempo de acceso a cualquiera de sus elementos es siempre igual por término medio.

La distinción entre ficheros directos y ficheros relativos depende, por el contrario, del criterio con el que se indentifican los registros individuales. Los ficheros directos propiamente dichos son aquellos en los que para apuntar a un registro se dan, en la práctica, las indicaciones de pista y sector del disco. En cambio, en los ficheros relativos, a cada registro se le asocia un número creciente relativo (primero, segundo, tercero, etc.) gracias al cual se puede apuntar directamente al registro. En este segundo caso corresponde al sistema operativo disco del ordenador (el DOS) convertir la indicación relativa del registro en los valores, a bajo nivel, de pista y sector.

Hablemos entonces de ficheros directos. Considerando que estos ficheros se suelen emplear solamente en los ordenadores personales maś grandes, que a menudo trabajan con el BASIC Microsoft, vamos a ver inmediatamente las instrucciones de este lenguaje. Entre los ordenadores caseros son pocos los que permiten los ficheros directos (entre las excepciones, el más significativo es el Commodore 64).

El punto crucial en la gestión de los ficheros directos es la preparación de la estructura del registro. Dijimos anteriormente que, en los ficheros directos, los registros tienen todos la misma longitud. Pero comprobar solamente que los datos tienen esta longitud no nos basta. Es preciso imaginar un registro como una caja en la que situamos distintos divisores: en cada compartimento (campo) se ponen los datos efectivos. Una vez que la caja esté preparada, es decir, cuando el programa haya llenado de datos los compartimentos, se enviará al fichero. El nombre correcto de esta caja es buffer, término que encontramos ya con anterioridad.

Un buffer es una zona de memoria temporal que se carga y se descarga de forma alternativa. Durante la fase de escritura del fichero el buffer se llena paso a paso, campo por campo, y luego todo el conjunto se transmite con rapidez al fichero (Figura 2).

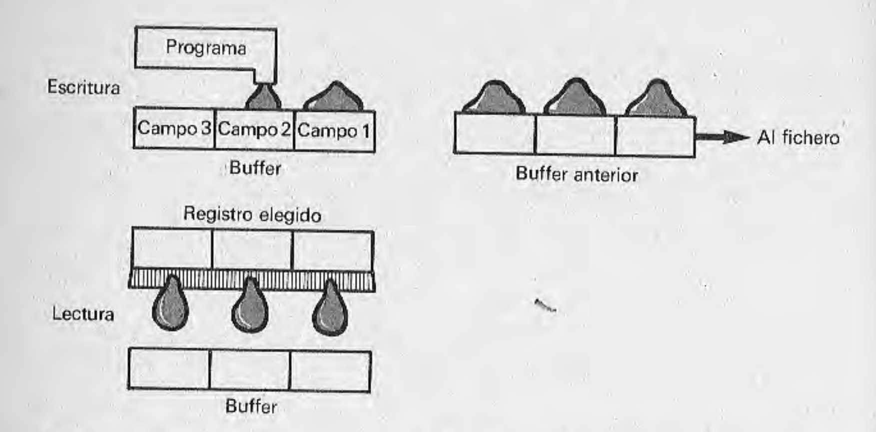


Figura 2.—Funcionamiento del buffer en la escritura y lectura de ficheros directos.

Por el contrario, durante la lectura, después de haber apuntado a un registro, éste se copia en el buffer, poniendo a disposición del programa sus datos, tal y como se ve en la Figura 2.

La instrucción del BASIC Microsoft que prepara y controla el buffer es la instrucción FIELD, y veremos a continuación cómo funciona.

FIELD, LSET, RSET y PUT

La instrucción FIELD determina una zona de memoria para el buffer, indispensable en la gestión de un fichero. Con FIELD se define exactamente la estructura del registro, indicando cuántos caracteres tiene de longitud y cuáles son sus campos. FIELD, que debe utilizarse tanto en escritura como en lectura, especifica cuántos y cuáles son los campos del registro y atribuye a cada uno un nombre de variable. Estos nombres de variable no representan variables ordinarias de las que estamos acostumbrados a emplear en BASIC, sino variables muy especiales a las que se le atribuye un valor mediante las instrucciones LSET o RSET; en la práctica son, más que variables, punteros dirigidos a los campos del registro.

FIELD #<canal>, <longitud1> AS <variable1> [,<long.2> AS <var.2>...]

LSET y RSET son instrucciones de carga del buffer (utilizadas solamente en la escritura del fichero). La primera, LSET ("left set") llena los campos del buffer a partir de la izquierda, como estamos acostumbrados para formar en columnas las cadenas, y la segunda, RSET ("right set"), realiza dicha operación desde la derecha, como se suele hacer con los números.

LSET <variable campo> = <variable>
RSET <variable campo> = <variable>

Quizá en este punto tenga las ideas más confusas que nunca. En realidad, el empleo de la instrucción FIELD y de las instrucciones LSET y RSET es muy sencillo. Un ejemplo servirá para aclararlo definitivamente (RSET se utiliza en más raras ocasiones, pero del mismo modo).

Supongamos que quiere escribir un fichero directo de nombres propios, cuyos registros están divididos en dos campos: nombre y apellido. Ante todo, debemos establecer cuál será la longitud de cada registro; si prevemos 30 caracteres para los nombres y 20 para los apellidos, los registros tendrán una longitud de 50 caracteres. Repetimos que esta delimitación de la longitud de los registros es característica de los ficheros directos, mientras que los ficheros secuenciales son libres de tener registros largos o cortos, indistintamente.

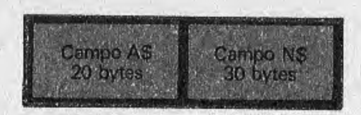
Veamos las dos primeras instrucciones:

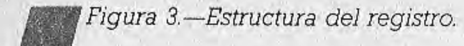
```
100 OPEN "R",#8,"NOMBRES PROPIOS",50
110 FIELD #8,30 AS N$,20 AS C$
```

OPEN abre un fichero relativo o directo ("R") con el número de canal 8, elegido libremente entre 1 y 255. El nombre del fichero es NOMBRES PROPIOS y 50 es la longitud total establecida para todos los registros. La instrucción OPEN para los ficheros secuenciales habría exigido en el lugar de R, una O para le escritura y una I para la lectura del fichero (la lectura y la escritura de los ficheros directos pueden ser simultáneas).

La instrucción FIELD, de la línea 110, indica que se refiere al fichero asociado al canal 8. Se podrían abrir simultáneamente varios ficheros, con tal de que tengan números de canal y nombres diferentes. Según definimos en FIELD, el registro está dividido en dos campos, llamados N$ y A$, el primero con una longitud de 30 caracteres y el segundo con 20 caracteres (Fig. 3).

En este punto, a título de ejemplo, supongamos que se piden a la entrada 10 nombres propios y que se quieren escribir en el fichero. Tendremos que definir instrucciones de entrada desde el teclado, comprobar que cada nombre no tenga más de 30 caracteres de longitud y cada apellido no más de 20 caracteres, y luego, a través de las instrucciones LSET, "asignaré" a cada campo




Figura 3.—Estructura del registro.

de los registros. Con la instrucción PUT# los valores de los campos (en el buffer) se pasan al fichero, como veremos en el siguiente apartado. Una vez concluidas todas las operaciones, se cerrará el fichero. Veamos el programa completo:

```
100 OPEN "R",#8,"NOMBRES PROPIOS",50
105 REM
110 FIELD #8,30 AS N$,20 AS A$
120 REM
130 FOR K=1 TO 10:REM 10 NOMBRES PROPIOS
140 INPUT "NOMBRE ";DN$
150 IF LEN(DN$)>30 THEN GOTO 140
160 INPUT "APELLIDO";DA$
170 IF LEN(DA$)>20 THEN GOTO 160
180 LSET N$=DN$:REM CAMPO DEL NOMBRE
190 LSET A$=DA$:REM CAMPO DEL APELLIDO
195 REM
200 PUT #8,K:REM ESCRITURA DEL REGISTRO K-ESIMO
210 NEXT K
220 REM
230 CLOSE #8
```

PUT# y GET#

Son las dos instrucciones que permiten escribir y leer un fichero directo. Ambas instrucciones, además de indicar el fichero al que se refieren (en el ejemplo, el asociado al canal núm. 8), tienen como único parámetro el número del registro que debe escribirse (PUT#) o leerse (GET#). Su sintaxis es:

PUT# <canal>, <número registro>
GET# <canal>, <número registro>

En este caso, le recordamos que la palabra reservada GET no tiene nada que ver con la GET que usan algunos ordenadores con los ficheros secuenciales. Esta GET# puede leerse así: extraer el registro n-ésimo del fichero abierto con el número de canal indicado y copiarlo en el buffer definido por la instrucción FIELD correspondiente. PUT# transfiere, por el contrario, el contenido del buffer al registro n-ésimo del fichero.

GET# es algo más sencilla de utilizar que PUT# por cuanto

que no requiere las instrucciones LSET o RSET. Después de que se haya leído un registro con GET#, las variables citadas en la instrucción FIELD pueden utilizarse inmediatamente como variables normales. Veámoslo escribiendo el programa de lectura del fichero NOMBRES PROPIOS.

```
100 OPEN "R",#2,"NOMBRES PROPIOS",50
105 REM
110 FIELD #2,30 AS N$,20 AS A$
120 REM
130 PRINT "QUE NOMBRE QUIERE LEER:1-10 (O FINAL)"
135 REM
140 INPUT NR:REM NUMERO DE REGISTRO
145 REM
150 IF NR=0 THE GOTO 200
160 IF NR<1 OR NR>10 THEN GOTO 130
165 REM
170 GET #2,NR:REM LECTURA DEL REGISTRO
175 REM
180 PRINT N$,A$:REM IMPRESION DEL REGISTRO
190 GOTO 130
195 REM
200 CLOSE #2:END
```

La instrucción PRINT en la línea 180 hace uso directamente de las dos variables N$ y A$ de la instrucción FIELD, sin pasar a través de LSET o RSET.

La instrucción FIELD hace referencia siempre a variables de cadena. Si queremos grabar números en un fichero directo, no podemos hacerlo directamente mediante variables numéricas, sino de forma indirecta a través de variables de cadena. Para convertir datos numéricos en una cadena se puede recurrir a la función STR$, y para lo contrario, a VAL, pero muchas versiones del BASIC, como la de Microsoft, proporcionan un conjunto de funciones especiales que hacen más inmediata la conversación de números en cadenas para grabarlos en los ficheros directos. Estas funciones son: MKI$, MKS$ y MKD$. La primera se refiere a los números enteros; la segunda, a los números de simple precisión (números con seis cifras significativas), y la tercera, a los números de doble precisión (con dieciséis cifras significativas).

Por el contrario, cuando se lee un fichero numérico se convierten las cadenas en números con CVI, CVS y CVD, que tratan a los números enteros, a los de simple precisión y a los de doble precisión, respectivamente.

Una vez más, con un pequeño ejemplo consideramos que se aclararán los conceptos:

```
80 REM *****************************
90 REM *    APERTURA DEL REGISTRO   *
95 REM *****************************
100 OPEN "R",#1,"NUMERICO",8
```

```
105 REM ***************************
110 REM * DEFINICION DEL REGISTRO *
115 REM ***************************
120 FIELD #1,4 AS X$,4 AS Y$
125 REM *******************
130 REM * ESCRIBO VALORES *
135 REM *******************
140 X1=32.345:Y1=23.00333
145 REM
150 RSET X$=MKS$(X1)
160 RSET Y$=MKS$(Y1)
165 REM ******************************
170 REM * INTRODUCCION EN EL FICHERO *
175 REM ******************************
180 PUT #1,1
185 REM **************************
190 REM * LECTURA DE LOS VALORES *
200 REM **************************
210 GET #1,1
220 REM *********************************
235 REM * TRANSFORMACION DE LOS VALORES *
240 REM *********************************
250 R1=CVS(X$)
260 R2=CVS(Y$)
270 PRINT R1,R2
280 REM **********************
285 REM * CIERRE DEL FICHERO *
290 REM **********************
300 CLOSE #1
```

El fichero tiene un solo registro con dos campos, en los que se escriben dos números reales de precisión simple, que ocupan 4 bytes cada uno, y luego se leen.

Una matriz numérica en fichero directo

En los apartados anteriores hemos visto el ejemplo de un fichero secuencial en el que se grababa una gran matriz numérica. Como prometimos entonces, vamos a resolver el mismo problema utilizando un fichero directo. En el mismo programa insertamos tanto la parte de escritura como la de lectura de la matriz.

También en este caso suponemos que la matriz es numérica y que tiene 100 filas y 50 columnas (línea 170), pero podría ser mucho más amplia para su grabación en disco, o bien tener dimensiones variables y determinadas por una sentencia INPUT anterior que sustituyera a la línea 170.

En la línea 200 se realiza la apertura del fichero con el parámetro "R" (fichero relativo o directo), el número de canal #1 y el nombre del fichero MAT. DIR. El número 4 (longitud del registro) es obligatorio y sabemos que determina la longitud de todos los registros del fichero. El valor dado (4) puede parecer pequeño, pues los ficheros tienen registros que suelen ser mucho más largos, pero el nuestro es solamente un ejemplo que se refiere a la

grabación de números reales (de simple precisión) que ocupan 4 bytes cada uno.

Inmediatamente después de la apertura del fichero se debe preestablecer la estructura del registro con la instrucción FIELD. En nuestro caso, el registro tiene un solo campo de 4 bytes de longitud. El nombre del campo es F$.

Antes de explicar el significado de la instrucción de la línea 240 y de la subrutina que comienza en la 520, vamos a ver el resto del programa. En las líneas 260 a 300 se pregunta si se pretende leer o escribir el fichero o si se quiere terminar. En el tercer caso, antes de ejecutar END, el fichero se cierra con CLOSE.

Si se quiere leer un elemento de la matriz se tendrán que proporcionar los números de fila y de columna (R y C). Para pasar de estos dos valores al de P, que indica la posición relativa del registro en el fichero, se utiliza la pequeña fórmula de la línea 360. No olvidemos que un fichero es siempre una estructura lineal, mientras que nuestra matriz es bidimensional. Para comprender cómo se calcula P basta pensar en el corte, línea por línea, de la matriz y en la posterior disposición de estas "rebanadas" en fila.

En la línea 370 GET# lee el registro P-ésimo, que se convierte en un número mediante la función CVS (CVS para los números reales con precisión simple, CVI para los enteros y CVD para los números reales de precisión doble).

Por el contrario, si se elige escribir un elemento de la matriz (subrutina 420) se calculará el puntero P como se vió antes (línea 450) y luego se transformará en cadena el elemento numérico con MKS$. Este valor se asignará al campo F$ mediante la instrucción LSET (línea 470). Finalmente, PUT# transfiere el registro P-ésimo al fichero.

Veamos ahora el significado de la línea 240. Para explicarnos mejor, supongamos que esta línea falta y que, antes de haber escrito ningún registro, se trata de leer alguno. A primera vista podríamos pensar que el ordenador nos debería avisar de que ese registro no existe o bien darnos como respuesta un valor nulo. En realidad, el ordenador intenta leer la parte de disco en donde debiera encontrarse el registro y lee los bytes que encuentra en ese punto (bytes que forman parte de cualquier otra grabación anterior). Esta situación absurda de lectura de datos "sucios" puede evitarse de una sola manera: preparando (inicializando) toda la zona del disco en la que se grabará el fichero.

Esta operación es precisamente la que desarrolla la subrutina en la línea 520. La función STRING$ en la línea 540 prepara una cadena de 4 caracteres "nulos" (ASCII 0). El doble bucle FOR NEXT apunta sucesivamente a todos los registros del fichero en los cuales escribe estos cuatro caracteres nulos. De este modo, si en cual-

quier momento tratara de leer un registro no escrito todavía, el ordenador le proporcionaría una cadena nula.

Para llamar a la subrutina de inicialización del fichero se recurre a la función LOF (LOF <canal>) que nos indica cuántos bloques de un fichero relativo están "comprometidos" con anterioridad. Si este valor es nulo (línea 240) quiere decir que no se hizo ninguna escritura, y entonces se salta a la línea 520.

```
100 REM *************************
110 REM *   ESCRITURA Y LECTURA  *
120 REM * DE UNA MATRIZ NUMERICA *
130 REM *   EN FICHERO DIRECTO   *
140 REM *************************
150 REM
160 REM
170 TR=100:TC=50:REM MATRIZ DE 100 X 50
180 REM
190 REM
200 OPEN "R",#1,"MAT.DIR",4
210 REM
220 FIELD #1,4 AS F$:REM BUFFER
230 REM
240 IF LOF(1)=0 THEN GOSUB 520
250 REM
260 INPUT "LECT.ESCRIT.FINAL (L,S,F)";Y$
270 IF Y$="L" THEN GOSUB 330
280 IF Y$="S" THEN GOSUB 420
290 IF Y$="F" THEN CLOSE #1:END
300 GOTO 260
310 REM
320 REM
330 REM ---- LECTURA DE LA MATRIZ ----
340 REM
350 INPUT "FILA Y COL.: ";R,C
360 P=(R-1)*TC+C
370 GET #1,P
380 PRINT "ELEMENTO: ";CVS(F$)
390 RETURN
400 REM
410 REM
420 REM ---- ESCRITURA DE LA MATRIZ ----
430 REM
440 INPUT "FILA Y COL.: ";R,C
450 P=(R-1)*TC+C
460 INPUT "ELEMENTO: ";E:REM NUMERO REAL
470 LSET F$=MKS$(E)
480 PUT #1,P
490 RETURN
500 REM
510 REM
520 REM ---- INICIALIZACION DE LA MATRIZ ----
530 REM
540 LSET F$=STRING$(4,CHR$(0))
550 REM
560 FOR R=1 TO TR
570 FOR C=1 TO TC
580 P=(R-1)*TC+C
590 PUT #1,P
600 NEXT C
```



```
610 NEXT R
620 REM
630 RETURN
```

Gestión de los clientes de un hotel

Después de haber hablado tanto de ficheros, le proponemos un programa, algo más completo, en el que se controla la presencia de los clientes de un hotel. El núcleo fundamental del programa es un fichero de acceso directo en el que se incluyen los datos más importante de cada cliente:

- nombre y apellido,
- fecha de llegada y
- precio de la habitación por día.

Para tener acceso a los registros individuales se utiliza como "clave" el mismo número de las habitaciones. En el supuesto de que el hotel tenga 100 habitaciones, utilizaremos los números del 1 al 100 para apuntar a los registros. Evidentemente este número se puede modificar cambiando la línea 170.

Las funciones desempeñadas por el programa son:

- registro de una nueva llegada, con indicación en el fichero del precio de la habitación, de la fecha de llegada y del nombre del huésped;
- impresión de la lista de nombres de los clientes alojados en el hotel;
- en caso de partida, cálculo del importe a pagar, y, finalmente,
- cierre del programa.

Las fechas de llegada y partida, proporcionadas en la forma de día y mes, se convierten en número del día del año (1-365) para facilitar el cálculo de los días de estancia. Por sencillez no se tiene en cuenta el año y, por consiguiente, el programa no funcionaría para clientes que estuvieran en el hotel "a caballo" entre dos años. Los días de estancia se calculan como diferencia entre la fecha de partida y la de llegada más uno (línea 4120). Ello es así para tener en cuenta que se paga la habitación aunque sólo se permanezca en el hotel desde la mañana a la tarde; si no está de acuerdo puede suprimirlo.

Como vimos en casos análogos, cuando el programa se utiliza por primera vez se debe inicializar el fichero directo. Así, en la línea 250 comprobamos con LOF(1) si el fichero no ha ocupado todavía ningún bloque; en tal caso saltamos a la subrutina de la línea 1000, que graba 100 registros con 38 caracteres nulos cada

uno (CHR$(0)). Dese cuenta que esto se hace con la variable IN$ que se definió como buffer en la línea 230, mientras que en la 220 usamos PR$, AR$ y NC$. Esto es totalmente correcto siempre que en una misma operación sólo usemos uno de los dos buffers creados. Cuando utilicemos normalmente el programa, además del fichero principal (HOTEL), se debe utilizar otro pequeño fichero (que hemos elegido secuencial) en el que indicar las habitaciones libres (HABITACIONES). Este fichero contiene una copia del vector LIB (100) en el que están indicadas las habitaciones libres con un 0, y las ocupadas, con un 1. Así, cuando se inicializa el fichero HOTEL se inicializará también el fichero HABITACIONES, todo él con valores 0 (línea 1170).

Cuando se ejecuta el programa, lo primero que sucede es que el fichero HABITACIONES se copia en LIB (líneas 270-340). A continuación aparece el menú, que pide al conserje del hotel la operación que desea efectuar.

Si es la llegada de un nuevo cliente, el programa busca, en el vector LIB, si hay habitación libre (líneas 2020-2120). CAM es el número de la habitación que ha de asignarse al cliente (con el número 0 todas las habitaciones están ocupadas). El bucle establecido en las líneas 2060-2080 busca el primer elemento nulo del vector LIB:

```
2070 IF LIB(K) = 0 THEN CAM = K:K = CA
```

En caso afirmativo, K se hace igual al número de la habitación, y luego, para salir correctamente del bucle, se establece K=CA (CA es el número total de las habitaciones).

Si no hay ninguna habitación libre, el programa imprimirá un aviso (línea 2100).

Una vez encontrada una habitación libre (CAM) se deben dar los datos de entrada: nombre del cliente, fecha de llegada y precio. La fecha se convierte en el día del año DA (subrutina de la línea 5000). La escritura en el buffer definido en la línea 220 con la instrucción FIELD se realiza con las tres instrucciones LSET (líneas 2230-2250). Las variables numéricas PREC y DA son convertidas en cadenas con MKS$. En la línea 2270 se graba en el fichero el registro de la habitación CAM e inmediatamente después (¡sería lamentable que un conserje lo olvidara!) se indica automáticamente que la habitación está ya ocupada LIB (CAM) = 1.

La subrutina para la lectura de los clientes alojados en el hotel es muy sencilla. Con la instrucción IF THEN en la línea 3030 se buscan las habitaciones ocupadas; a partir de ellas se lee el registro con GET# y luego se imprimen los datos.

Cuando se va un cliente, a través de su número de habitación PA se lee el registro, a partir del cual se obtiene el precio de la

habitación y el día de llegada. Con éstos y la fecha de partida que se solicita se calcula el importe total a pagar (línea 4180). Inmediatamente después se pone a "cero" el registro (línea 4250-4260) y se indica que está libre la habitación correspondiente en el vector LIB.

El cierre del programa copia el vector LIB en el fichero HABITACIONES para permitir el correcto control del hotel en la siguiente ejecución del programa, y luego cierra los dos ficheros HOTEL y HABITACIONES.

Damos a continuación las variables utilizadas en el programa:

● CA	número de habitaciones del hotel,
● PR$, PREC	precio de la habitación,
● DD,MM	día y mes,
● AR$,DA	día del año,
● NC$, NOM$	nombre y apellidos del cliente,
● IN$	cadena nula de inicialización,
● LIB	vector de ocupación de las habitaciones del hotel,
● CAM	primera habitación libre,
● PA	habitación del huésped,
● DE	días de estancia,
● PAG	importe del servicio.

```
100 REM ***********************
110 REM *  RECEPCION DE HOTEL  *
120 REM ***********************
130 REM
140 REM
150 REM
160 REM
170 CA=100:REM NUMERO DE HABITACIONES DEL HOTEL
180 DIM LIB(CA):REM VECTOR HABITACIONES LIBRES
190 REM
200 OPEN "R",#1,"HOTEL",38
210 REM
220 FIELD #1,4 AS PR$,4 AS AR$,30 AS NC$
230 FIELD #1,38 AS IN$
240 REM
250 IF LOF(1)=0 THEN GOSUB 1000
260 REM
270 REM ---- PREPARACION DE VECTOR LIB ----
280 REM ---- DE LAS HABITACIONES LIBRES ---
300 OPEN "I",#2,"HABITACIONES"
310 REM
320 FOR J=1 TO CA
330 INPUT #2,LIB(J)
340 NEXT J
350 REM
360 REM
370 REM ---- MENU DE RECEPCION ----
380 REM
390 CLS: REM BORRADO DE LA PANTALLA
400 REM
```

```
410 PRINT "NUEVA LLEGADA --------- N"
420 PRINT "CLIENTES DEL HOTEL --- C"
430 PRINT "PARTIDA DEL CLIENTE -- P"
440 PRINT "FINAL DE OPERACIONES - F
450 REM
460 INPUT "ELIJA SU OPCION";R$
470 REM
480 IF R$="N" THEN GOSUB 2000
490 IF R$="C" THEN GOSUB 3000
500 IF R$="P" THEN GOSUB 4000
510 IF R$="F" THEN GOSUB 8000
520 GOTO 460
530 REM
540 REM
550 REM
1000 REM ---- SUBRUTINA DE INICIALIZACION ----
1010 REM
1020 REM --- PREPARACION DEL FICHERO HOTEL ---
1030 REM
1040 LSET IN$=STRING(38,CHR$(0))
1050 REM
1060 FOR K=1 TO CA
1070 PUT #1,K
1080 NEXT K
1090 REM
1100 REM --- PREPARACION DEL FICHERO HABITACIONES ---
1110 REM
1120 LIBRE=0
1130 REM
1140 OPEN "O",#2,"HABITACIONES"
1150 REM
1160 FOR J=1 TO CA
1170 PRINT#2,LIBRE
1180 NEXT J
1190 REM
1200 CLOSE #2
1210 REM
1220 RETURN
1230 REM
1240 REM
1250 REM
2000 REM ---- SUBRUTINA DE NUEVA LLEGADA ----
2010 REM
2020 REM --- BUSQUEDA DE HABITACION LIBRE ---
2030 REM
2040 CAM=0:REM NUMERO DE LA PRIMERA HABITACION
2050 REM   LIBRE  (0  TODO OCUPADO)
2060 FOR K=1 TO CA
2070 IF LIB(K)=0 THEN CAM=K:K=CA
2080 NEXT K
2090 REM
2100 IF CAM=0 THEN PRINT "TODO OCUPADO!":RETURN
2110 REM
2120 PRINT "HABITACION ";CAM;" LIBRE"
2130 REM
2140 INPUT "NOMBRE Y APELLIDOS DEL CLIENTE (30 CAR. MAX.):";NO
2150 IF LEN(NOM$)>30 THEN GOTO 2140
2160 INPUT "FECHA DE LLEGADA (DD/MM):";DD,MM
2170 INPUT "PRECIO DE LA HABITACION:";PREC
2180 REM
2190 GOSUB 5000:REM CONVERSION DE LA FECHA
2200 REM
2210 REM --- ESCRITURA EN EL BUFFER ---
2220 REM
2230 LSET PR$=MKS$(PREC)
```



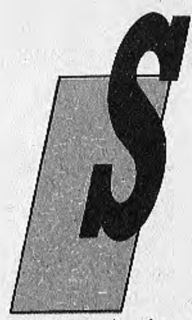
```
2240 LSET AR$=MKS$(DA)
2250 LSET NC$=NOM$
2260 REM
2270 PUT #1,CAM:REM CAM ES TAMBIEN EL PUNTERO DEL REGISTRO
2280 LIB(CAM)=1:REM HABITACION OCUPADA
2290 REM
2300 RETURN
2310 REM
2320 REM
2330 REM
3000 REM ---- SUBRUTINAS DE CLIENTES DEL HOTEL ----
3010 REM
3020 FOR K=1 TO CA
3030 IF LIB(K)=1 THEN GOTO 3070
3040 GET #1,K:REM LECTURA DEL REGISTRO
3050 REM
3060 PRINT "D.";NC$;"  HABITACION ";K
3070 NEXT K
3080 REM
3090 REM
3100 REM
4000 REM ---- SUBRUTINA DE PARTIDA ----
4010 REM
4020 INPUT "QUE HABITACION QUEDA LIBRE";PA
4030 INPUT "FECHA DE PARTIDA (DD/MM):"DD,MM
4040 GOSUB 5000:REM DIA DEL AÑO
4050 REM
4060 REM LECTURA DEL FICHERO
4070 REM
4080 GET #1,PA
4090 REM
4100 REM DIAS DE ESTANCIA
4110 REM
4120 DE=DA-CVS(AR$)+1
4130 REM
4140 PRINT "D.",NC$;"HA ESTADO ";DE;" DIAS"
4150 REM
4160 REM PAGO DEL SERVICIO
4170 REM
4180 PAG=DE*CVS(PR$)
4190 REM
4200 PRINT "EL PRECIO DE LA HABITACION ES:";PAG
4210 PRINT "GRACIAS Y ADIOS"
4220 REM
4230 LIB(PA)=0:REM DEJAR LIBRE LA HABITACION
4240 REM
4250 LSET IN$=STRING$(38,CHR$(0))
4260 PUT #1,PA
4270 REM
4280 RETURN
4290 REM
4300 REM
4310 REM
5000 REM SUBRUTINA DE CONVERSION DE LA FECHA AL DIA DEL AÑO
5010 REM
5020 DA=DD
5030 FOR J=1 TO MM
5040 READ MM
5050 DA=DA+MM
5060 NEXT J
5070 REM
5080 DATA 0,31,28,31,31,30,31
5090 DATA 30,31,31,30,31,30
5100 REM
5110 RETURN
```

```
5120 REM
5130 REM
5140 REM
8000 REM ---- CIERRE DEL PROGRAMA ----
8010 REM
8020 REM ---- ALMACENAMIENTO DEL VECTOR ----
8030 REM --- DE LAS HABITACIONES LIBRES ---
8040 OPEN "O",#2,"HABITACIONES"
8050 REM
8060 FOR J=1 TO CA
8070 PRINT #2,LIB(J)
8080 NEXT J
8090 REM
8100 CLOSE #2:REM CIERRE DEL FICHERO DE LAS HABITACIONES
8110 REM
8120 CLOSE #1:REM CIERRE DEL FICHERO DEL HOTEL
8130 REM
8140 PRINT "FINAL DEL TRABAJO"
8150 REM
8160 END
```


CAPITULO VI

INSTRUCCIONES GRAFICAS Y ANIMACION

¿Por qué los gráficos?



Sabemos que los ordenadores dialogan con nosotros fácilmente mediante los caracteres alfabéticos y numéricos normales. Un programa se escribe como si fuera una carta a un amigo y el ordenador responde con mensajes que se leen como si estuvieran escritos en un periódico. Junto a este "canal" alfanumérico podemos comunicarnos también con otros medios, como son la imagen y el sonido.

Las imágenes son todo lo que el ojo ve y la mano puede trazar, y los sonidos son todo lo que oímos y que nuestra garganta u otro instrumento puede generar. Cuando utilizamos las imágenes para comunicarnos con el ordenador hacemos uso de gráficos. En el segundo caso no existe todavía un término específico, salvo hablar de música con ordenador o síntesis vocal.

Con los gráficos del ordenador, el concepto de proceso de datos se amplía y asume aspectos muy diferentes y fascinantes. Si proporcionamos a un ordenador imágenes, por ejemplo el dibujo de un automóvil, nos puede responder con otros dibujos; como la vista frontal o lateral del mismo automóvil y, al mismo tiempo, nos puede calcular el área de una de sus superficies. Para lograr esto debemos disponer de programas que sean capaces de realizar el proceso de datos gráficos; estos programas constituyen lo que se denomina el software gráfico.

Los gráficos permiten emplear el ordenador de un modo completamente nuevo. Junto a los periféricos tradicionales encontramos los adaptados a la entrada y a la salida de imágenes: plotter,

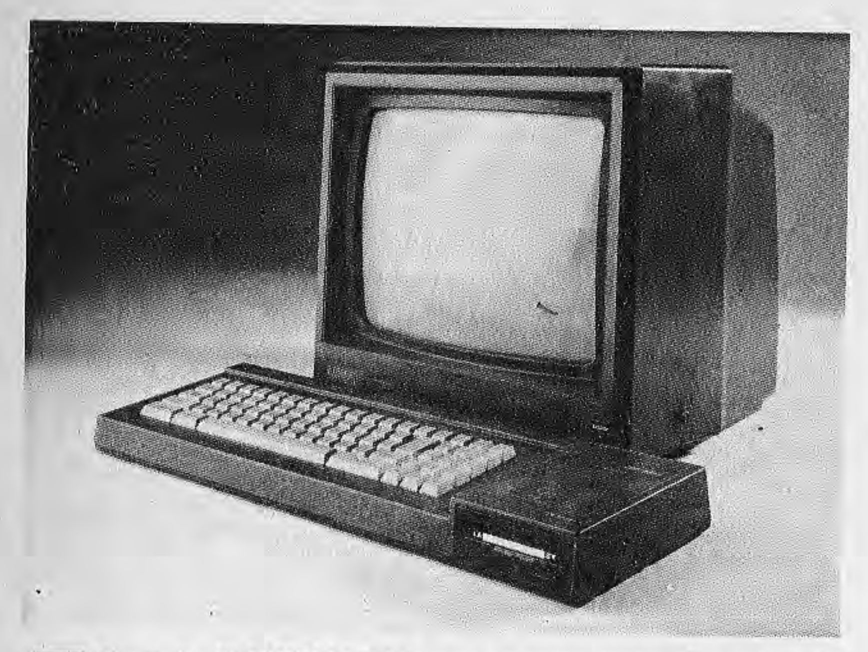


Figura 1.—El AMSTRAD-664, capaz de comunicarse mediante gráficos y sonido (en estéreo).

tablillas gráficas, lápices ópticos, ratón, joystick, track ball, etc. Una versión diferente, más reducida, pero muy importante, de los gráficos es la que se refiere a los ordenadores personales.

Para efectuar gráficos de tipo profesional es necesario disponer de ordenadores de dimensiones medias o grandes, de periféricos gráficos muy costosos y de un software gráfico un tanto complejo. Un requisito muy importante es el de la velocidad de cálculo de la CPU, que debe ser muy elevada. Para comprender esto considere que, con frecuencia, para situar un solo punto de una imagen se deben efectuar primero cálculos largos y complejos. En una imagen constituida por miles de puntos, si el cálculo de cada uno de ellos no se realiza en un tiempo muy breve, no se conseguirá ningún resultado práctico.

Es precisamente ésta la limitación de los ordenadores personales en relación con los gráficos profesionales: son demasiado lentos y no tienen suficiente memoria central, pero, sobre todo, el precio de los buenos periféricos gráficos es desproporcionado con respecto al de cualquier ordenador personal.

Por todo esto, con los ordenadores personales han nacido unos nuevos tipos gráficos. Unos gráficos económicos y sencillos, pero de gran utilidad, que les coloca a la altura de sus hermanos

mayores. Los gráficos de los ordenadores personales son una forma de presentar los resultados de un programa y constituyen un modo para hacer más agradable la visualización de los datos, pero son también un medio de realizar programas de juegos ("los denominados videojuegos") o para hacer más cercanos o amigables muchos programas que por sí solos serían aburridos, tales como los programas didácticos.

Esto es una necesidad y una exigencia de los ordenadores personales y justifica la existencia, tanto en el BASIC como en otros lenguajes para ordenadores personales, de instrucciones gráficas que no existen en las versiones para grandes ordenadores de los mismos lenguajes. Se trata de instrucciones que permiten colorear la pantalla, trazar puntos, rectas, curvas o programar figuras (los denominados "sprites") que se mueven y animan con facilidad en la pantalla.

Como confirmación de que los gráficos "personales" tienen su papel autónomo basta observar que los ordenadores más ricos en estas opciones son precisamente los ordenadores personales de la gama más baja y de mayor éxito comercial. Los gráficos en ordenadores personales "mayores" no se proporcionan para "jugar", sino para enriquecer la visualización de resultados en los programas profesionales.

El "sketchpad" de Ivan Sutherland

Cualquier alusión histórica a los gráficos nos lleva a muchos años atrás. La necesidad de comunicarse con el ordenador, no solamente en el modo alfanumérico, sino también con la posibilidad de efectuar algoritmos en conjuntos geométricos, se sintió inmediatamente por los primeros usuarios de los ordenadores.

El primer sistema gráfico fue desarrollado en el año 1963 en el prestigioso MIT de Boston por Ivan Sutherland. El sistema, que se llamaba SKETCHPAD (tablero de dibujo), era interactivo, permitía dibujar en la pantalla figuras elementales y estaba provisto del primer lápiz óptico ("light pen"). Mediante mandos situados cerca de la pantalla (Fig. 2) era posible obtener figuras elementales, que luego se desplazaban y se agrupaban en la pantalla con el lápiz óptico.

El sistema Sketchpad fue el primer ejemplo de "trabajo con gráficos" y requería fundamentalmente una pantalla gráfica, un lápiz óptico y el software necesario. Junto a este tipo de hardware se desarrollaron, a partir de los años cincuenta, otros periféricos. El más importante fue el plotter digital (digitalizador), gracias al cual se pueden hacer unos maravillosos y perfectos dibujos constituidos por puntos.

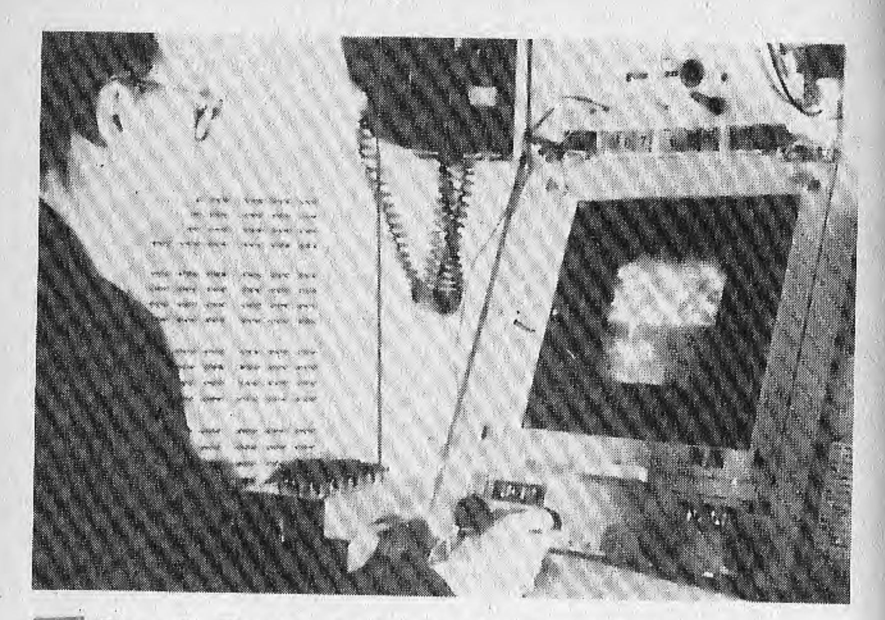


Figura 2.—El ordenador Sketchpad de Ivan Sutherland.

Programación de gráficos en los ordenadores personales

Todas las versiones del BASIC permiten realizar gráficos, pero las instrucciones disponibles para dicha tarea son muy diferentes de un ordenador a otro. Si quisiéramos convertir un programa de Apple que controle ficheros en un programa para el Commodore encontraríamos, casi con toda seguridad, las correspondencias entre las instrucciones de los dos ordenadores. Dicho de otro modo, el algoritmo concebido para controlar ficheros con el ordenador de Apple sigue siendo el mismo en el caso del Commodore y sólo hay que volverlo a escribir haciendo uso de instrucciones con diferentes sintaxis. Con los gráficos esto es prácticamente imposible. Es precisamente el algoritmo gráfico el que cambia por completo de un ordenador Apple a un Commodore o de un Spectrum a un IBM PC o a cualquier otro ordenador.

Con un poco de buena voluntad, como hicimos en otros casos, es posible determinar algunos métodos comunes para hacer gráficos. Fundamentalmente existen tres:

- empleo de caracteres semigráficos,
- creación de sprites (objetos móviles),
- control directo de los puntos de pantalla ("pixels").

Caracteres semigráficos

Se trata de la programación gráfica más sencilla y más común, pero también la que proporciona los resultados menos atractivos. El conjunto de los caracteres utilizados por el ordenador se amplía con nuevos caracteres que representan figuras elementales tales como trazos verticales, horizontales, diagonales, cuadrados rellenos, etc. (Fig. 3). Estos caracteres están codificados en ASCII, tal como los alfanuméricos, y se tratan de un modo idéntico. Para trazar un dibujo se hace uso de instrucciones PRINT que imprimen cadenas con estos caracteres.

Muchas veces se trazan dibujos elementales sin utilizar ni siquiera los caracteres semigráficos, ayudados solamente por caracteres simples, tales como el asterisco, el signo menos o el signo de exclamación. Un dibujo de esta clase es realizable por cualquier ordenador personal, y éste es el único caso de compatibilidad completa entre ordenadores.

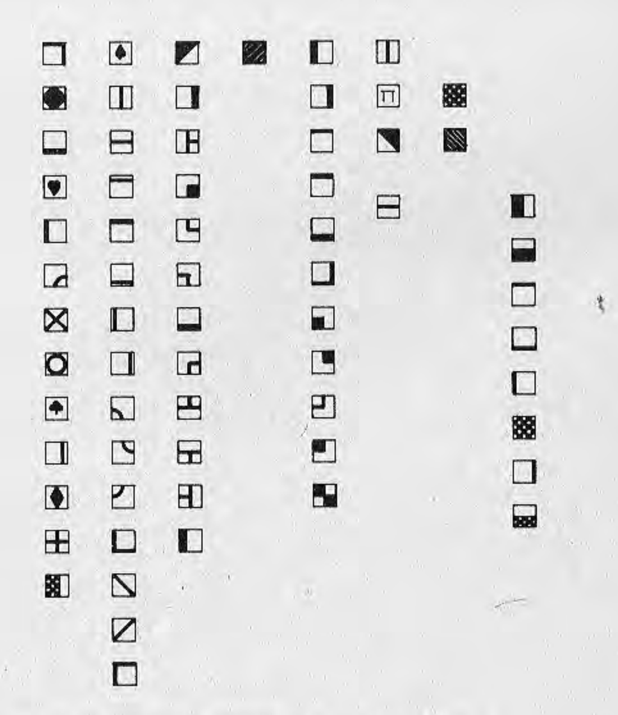


Figura 3.—Conjunto de caracteres semigráficos.

Por ejemplo, con las instrucciones:

```
PRINT "********"
PRINT "*      *"
```

se puede dibujar un cuadrado



Si se quisiera dibujar un círculo o una curva más compleja es fácil imaginar que los resultados dejarían bastante que desear.

El primer tipo de gráficos tiene una limitación notable, y es que permite trazar solamente dibujos simples y modulares, en función del conjunto de caracteres semigráficos que posea el ordenador. En compensación, es muy fácil de programar porque, como dijimos anteriormente, emplea las instrucciones PRINT normales.

Sprites

Los sprites son un paso adelante en la programación gráfica y permiten obtener resultados muy satisfactorios. Un sprite es una pequeña imagen móvil que se construye por el programador. Cada sprite definido en un programa tiene su nombre (como una variable) y puede visualizarse a voluntad, desplazarse y ampliarse en la pantalla con las instrucciones adecuadas. Con la alternancia de sprites diferentes se pueden conseguir efectos de movimiento, y en el caso límite, crear algo muy similar a las imágenes de los videojuegos. Los sprites se definen al comienzo del programa mediante una matriz (Fig. 4). Cada cuadrado elemental de la matriz define el estado (activado o apagado) de un punto de la pantalla.

El número máximo de sprites, su tamaño y otras características dependen del ordenador concreto. En general, cada sprite se mueve en un plano propio (como si fueran láminas plásticas superpuestas), cada uno con su prioridad. Así, por ejemplo, los equipos MSX admiten 32 planos más el fondo (Fig. 5).

Gráficos por puntos

El tercer tipo de gráficos es el más complejo, pero también el más perfeccionado. En este caso, el programador puede con-

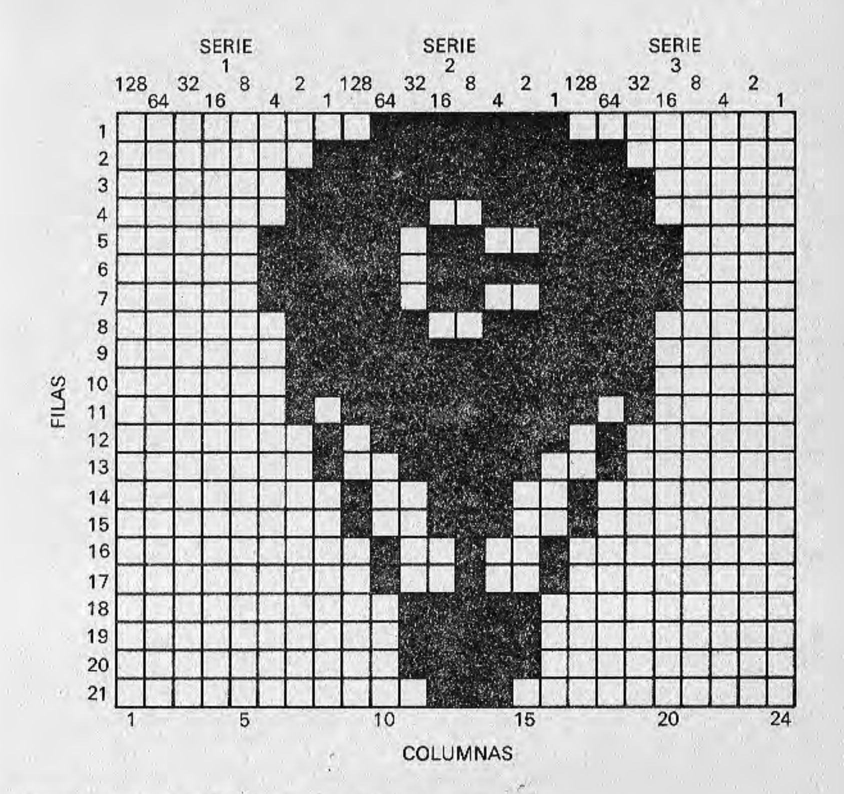


Figura 4.—Matriz de creación de un sprite.

trolar cada elemento de imagen individual de la pantalla (pixel), eligiendo también el color que se le ha de dar.

Los elementos de imagen, también denominados pixels (por la contracción de la denominación inglesa "picture cell") son los puntos unitarios de una imagen digital. Un pixel equivale a un bit: punto luminoso encendido o apagado. Cuantos más pixels existan en una pantalla (su número no depende del software, sino del hardware), tanto más definida y agradable será la imagen obtenida. Actualmente, en los sistemas gráficos profesionales se pueden obtener imágenes de calidad superior a la de una buena fotografía.

Muchos de estos sistemas tienen 1024 × 1024 pixels, que son más de un millón en la pantalla completa y cada uno puede iluminarse con cualquier color de entre 16 millones de colores. No piense que este elevadísimo número de colores es desproporcionado, pues comprende los matices de intensidad de los colores de base.

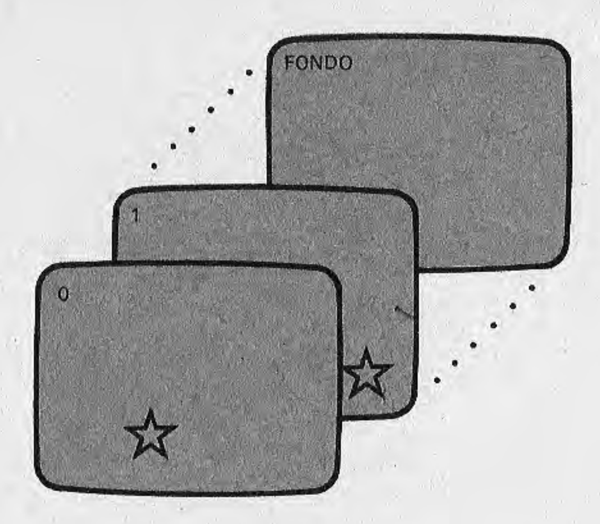


Figura 5.—Cada sprite es como si se moviera en un plano distinto de la pantalla. Según el ordenador, se admiten más o menos "planos".

Los ordenadores personales trabajan con un número de pixels inferior (por ejemplo, el Commodore 64 tiene 320×200 = 64000 pixels). Si recordamos que las pantallas más usuales de presentación visual no tienen "memoria" y que su imagen debe ser continuamente "refrescada", ello significa que los pixels de una imagen deben almacenarse en una memoria RAM que funciona como un buffer (Fig. 5). Esta zona de memoria RAM o "página gráfica", contiene *un mapa* de los puntos de la pantalla, y por este motivo, se denomina también memoria de "mapeado" de bits (en inglés, bit-mapping memory). Puesto que cada byte de memoria contiene 8 bits, se puede comprobar que el Commodore 64 utiliza para su página gráfica 8 Kbytes (64000/8=8 K.).

Cuanto mayor sea el número de pixels, tanto más grande debe ser la cantidad de memoria RAM necesaria para almacenar la imagen. Se comprende, pues, por qué los ordenadores personales no pueden tener muchos pixels. Si además son gráficos en color, junto a cada bit que indica un pixel activo debemos dar también la información de color. Para elegir entre los 8 colores habituales, en la mayor parte de los ordenadores personales son necesarios 3 bits (3 bits permiten seleccionar uno de entre ocho casos posibles). En el caso de una página gráfica de 8 Kbytes se deben añadir, desde el punto de vista teórico, otros 24 Kbytes para el color.

Los gráficos por puntos, que también se denominan de *alta resolución,* hacen uso de instrucciones que controlan cualquier punto individual de la pantalla, estando definido cada punto por sus coordenadas X,Y. Con frecuencia, para fines prácticos, existen instrucciones que hacen trazar una recta entre dos puntos (un vector) o bien un círculo o una elipse. Otras instrucciones son capaces de rellenar con color una curva cerrada.

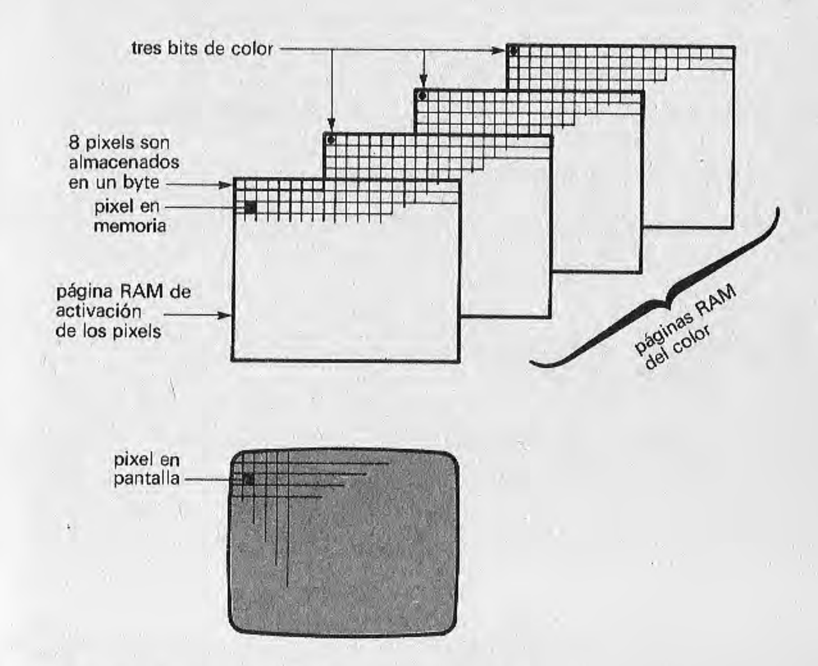


Figura 6.—Páginas gráficas de un ordenador.

Un ejemplo gráfico

No es fácil dar ejemplos de gráficos sin caer en la banalidad o sin recurrir a instrucciones demasiado específicas (y "oscuras"). No obstante, consideramos que para hacer algo más concreto cuanto hemos dicho, es interesante mostrar un programa, escrito para el ordenador Apple II, en el que se dibuja un cuadrado que se mueve rebotando de un extremo a otro de la pantalla. Con respecto a las tres clasificaciones antes aludidas, este ejemplo entra dentro de la tercera categoría, la de gráficos de alta resolución.

El lado L del cuadrado, que queremos hacer desplazar, se solicita como dato a la entrada (línea 180). Puesto que trabajamos

en alta resolución, este valor corresponde al número de pixels utilizados por el ordenador para dibujar cada lado del cuadrado. La limitación a 100 se eligió para tener un cuadrado aceptable, considerando que existen 280 × 160 pixels en las coordenadas X e Y de la pantalla del Apple II. La instrucción HGR en la línea 220 llama la "página 1" de los gráficos de alta resolución del ordenador Apple II (en realidad, este ordenador tiene dos páginas gráficas de alta resolución). La instrucción HPLOT es la que traza el cuadrado como sucesión de cuatro segmentos L consecutivos:

```
HPLOT   VERTICE1  TO  VERTICE2
        TO VERTICE3  TO  VERTICE4
        TO VERTICE1
```

La repetición final del vértice 1 sirve para cerrar el cuadrado. El programa elige al azar la coordenada Y del primer vértice del cuadrado, el situado en la parte superior izquierda, mientras que la X inicial es nula. Los incrementos DTX y DTY, para el desplazamiento del cuadrado son elegidos al azar en las líneas 260 y 270. Las instrucciones 310 y 320 sirven para controlar que el cuadrado no salga de la pantalla, sino que rebote; en efecto, apenas toca los bordes, los incrementos DTX y DTY cambian de signo y así se restan en lugar de sumarse, o viceversa.

En las líneas 340 y 410 se elige el color de trazo. Con HCOLOR = 3 el cuadrado se traza en blanco, mientras que con el valor 0, el color es negro como el fondo de la pantalla. En la práctica, el cuadrado se dibuja primero en blanco y luego en negro y ello equivale a borrarlo, pero después de algunos segundos, que son suficientes para que nuestros ojos puedan verlo. Repitiendo este doble trazo se obtiene un efecto visual de movimiento. El pequeño bucle de espera en la línea 390 sirve para hacer más suave el movimiento.

```
100 REM ***************
110 REM *  UN CUADRADO *
120 REM *  QUE REBOTA  *
130 REM ***************
140 REM
150 REM
160 TEXT:HOME
170 REM
180 INPUT "LADO DEL CUADRADO (1-100): ";L
190 IF L<1 OR L>100 THEN GOTO 180
200 REM
210 REM
220 HGR:REM ACTIVACION DE LA ALTA RESOLUCION
230 REM
240 X=0: Y=RND(1)*(150-L)
250 REM
260 DTX=RND(1)*5
270 DTY=RND(1)*5
```

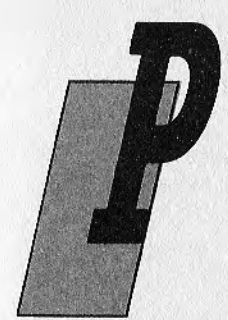


```
280 REM
290 REM --- INICIALIZACION DEL MOVIMIENTO ---
300 REM
310 IF X+DTX+L>279 OR X+DTX<0 THEN DTX=-DTX
320 IF Y+DTY+L>159 OR Y+DTY<0 THEN DTY=-DTY
330 REM
340 HCOLOR=3:REM TRAZO DE COLOR BLANCO
350 REM
360 X=X+DTX:Y=Y+DTY
370 HPLOT X,Y TO X+L,Y TO X+L,Y+L TO X,Y+L TO X,Y
380 REM
390 FOR K=1 TO 50 :NEXT K: REM BUCLE DE RETARDO
400 REM
410 HCOLOR=0:REM TRAZO EN COLOR NEGRO (BORRADO)
420 REM
430 HPLOT X,Y TO X+L,Y TO X+L,Y+L TO X,Y+L TO X,Y
440 REM
450 GOTO 310
```

Como ya habrá observado, el diseño de gráficos, aunque no es difícil, sí resulta complicado, especialmente por la diversidad de instrucciones de cada ordenador. Pero no se preocupe demasiado: habrá más libros de nuestra biblioteca que vuelvan sobre estos temas.

APENDICE A

RESUMEN DE LAS INSTRUCCIONES ESTUDIADAS



ara facilitar al lector la rápida solución de cualquier duda sobre la sintaxis de las instrucciones vistas en este volumen de la BBI hemos confeccionado un pequeño resumen-diccionario, que esperamos le resulte útil.

Además de definir la función de la instrucción y su sintaxis, incluye unos cuantos ejemplos de muestra. En ningún caso incluimos el número de línea, pues damos por sentado que el lector sabe que si la instrucción forma parte de un programa debe llevarlo y, en caso contrario (ejecución directa), no.

En las definiciones que siguen usaremos los símbolos.

[ ] elemento opcional
{ } posibilidades alternativas
<> deben ser determinados por el usuario.

CLOSE

Cierra el canal lógico abierto mediante un OPEN

CLOSE #<n.º canal>

CLOSE #1

CVI, CVS, CVD

Convierten una cadena en el número que representa (entero, de simple o de doble precisión, respectivamente).

CVI<variable de cadena>
CVS<variable de cadena>
CVD<variable de cadena>

E = CVI (ENTERO$)
S = CVS (SIMPLEPRE$)
D = CVD (DOBLEPRE$)

DEF FN

Crea una nueva función, que se podrá usar allí donde se incluya esta definición

DEF FN (<parámetro mudo 1> [,...]) =

DEF FN (H) = 2*H+5
DEF FN (R) = 2*π*R

FIELD

Determina una zona de memoria para buffer de un fichero directo, especificando longitudes de cada campo y las variables asociadas.

FIELD #<nº canal>, <longitud 1> AS <variable de cadena 1> [,<long 2> AS <var 2>,...]

FIELD #1, 27 AS NOMBRE$, 10 AS DNI$

GET#

Carga en las variables dadas en FIELD los valores del registro seleccionado de un fichero directo

GET #<nº canal>, <nº registro>

GET #1, I
GET #3, PP+1

GOSUB

Cede el control a una subrutina que comienza en el número de línea especificado

GOSUB <nº de línea>

GOSUB 5000

INPUT#

Lee los campos del siguiente registro de un fichero secuencial, cargando sus valores en las variables explicitadas.

INPUT#<nº canal>, <variable de cadena 1> [,<var 2>...]

INPUT#5, P$, VALOR$

LINE INPUT#

Lee un registro completo de un fichero secuencial hasta el carácter RETURN

LINE INPUT# <nº canal>, <variable de cadena>

LINE INPUT#1, L$

LOF

Indica los bloques usados de un fichero directo

LOF (<nº canal>)

A = LOF (1)

LSET

Carga las variables de campo del buffer de un fichero directo de izquierda a derecha

```
LSET <variable campo> = <variable>
```

LSET NOMBRE$ = PEPE$

MKI$, MKS$, MKD$

Cargan en una cadena el valor de un número entero, de simple o de doble precisión, respectivamente

```
MKI$ (<variable>)
MKS$ (<variable>)
MKD$ (<variable>)
```

PEPE$ = MKS$ (DNI)
A$ = MKI$ (ENTERO)

ON GOSUB

Salta a la subrutina cuya línea de comienzo está en la posición de la lista cuyo número coincide con el valor que toma la variable (de 1 en adelante)

```
ON <variable> GOSUB <nº línea SUB1>, [<SUB2>,...]
```

ON POS GOSUB 1000, 2000, 3000

ON GOTO

Salta a la línea cuya posición en la lista coincide con el valor que toma la variable (de 1 en adelante)

```
ON <variable> GOTO <nº línea 1>,[,<nº 2>... ]
```

ON LIN GOTO 100, 200, 300, 400

OPEN

Establece la apertura de un canal lógico. Hay muchas sintaxis posibles. Las más comunes son:

- Ficheros directos

```
OPEN "R",# <nº canal>, "<nombre>", <longitud registro>
OPEN "<nombre>", AS[#] <nº canal>, LEN=<longitud registro>
```

OPEN "R", #1, "DISCO.1", 150

- Ficheros secuenciales

```
OPEN "<modo>", # <nº canal>, "<nombre>"
OPEN "<nombre> FOR "<modo>" AS <nº canal>
```

OPEN "I", #1, "PEPE"
OPEN "O", #2, SALIDA$

PRINT#

Escribe en el siguiente registro de un fichero secuencial el contenido de la lista de variables

```
PRINT# <nº canal>, <variable de cadena 1>,[;<var2>...]
```

PRINT#1, NOM$";";APELL$
PRINT#3, AB

PUT#

Escribe en el registro dado de un fichero directo el valor de las variables definidas en FIELD

```
PUT#<nº canal>, <nº registro>
```

PUT#5J

RND

Obtiene un número pseudoaleatorio. Si la base es nula repite el último número

RND (nº entero)

VALOR = RND(5)
REPITE = RND(0)

RSET

Carga las variables de campo del buffer de un fichero directo de derecha a izquierda

RSET <variable campo> = <variable>

RSET VAR$ = JUAN$

RTS

Devuelve el control al programa que llamó a la subrutina justo en la línea siguiente a aquella desde la que se realizó la llamada

RTS

APENDICE B

TABLAS DE CONVERSION

Si usted ha logrado el listado de un programa que le interesa, pero que está escrito en una versión del BASIC que no es la que tiene su propio ordenador, o bien simplemente desea ver de qué otras formas se pueden expresar determinadas instrucciones, las tablas que adjuntamos le serán de gran utilidad.

Están incluidas en ellas las equivalencias de todas las instrucciones vistas en los dos volúmenes del "Y llegó el BASIC", ordenadas alfabéticamente, para los siguientes equipos:

APPLE
COMMODORE 64
MSX
IBM-PC
QL
SPECTRAVIDEO
SPECTRUM
AMSTRAD 664
MACINTOSH

	ABS(<NUMERO>)	ACS(<X>)	ASC(<CADENA DE CARACTERES>)	ASN(<X>)	ATN(<X>)	CHR$(<NUMERO DE CODIGO ASCII>)
APPLE	ABS(<NUMERO>)	——	ASC(<CADENA DE CARACTERES>)	——	ATN(<X>)	CHR$(<NUMERO DE CODIGO ASCII>)
COMMODORE 64	ABS(<NUMERO>)	——	ASC(<CADENA DE CARACTERES>)	——	ATN(<X>)	CHR$(<NUMERO DE CODIGO ASCII>)
MSX	ABS(<NUMERO>)	——	ASC(<CADENA DE CARACTERES>)	——	ATN(<X>)	CHR$(<NUMERO DE CODIGO ASCII>)
IBM - PC	ABS(<NUMERO>)	——	ASC(<CADENA DE CARACTERES>)	——	ATN(<X>)	CHR$(<NUMERO DE CODIGO ASCII>)
QL	ABS(<NUMERO>)	ACS(<X>)	CODE(<CADENA DE CARACTERES>)	ASN(<X>)	ATN(<X>)	CHR$(<NUMERO DE CODIGO ASCII>)
SPECTRAVIDEO	ABS(<NUMERO>)	——	ASC(<CADENA DE CARACTERES>)	——	ATN(<X>)	CHR$(<NUMERO DE CODIGO ASCII>)
SPECTRUM	ABS(<NUMERO>)	ACS(<X>)	CODE(<CADENA DE CARACTERES>)	ASN(<X>)	ATN(<X>)	CHR$(<NUMERO DE CODIGO ASCII>)
AMSTRAD 664	ABS(<NUMERO>)	——	ASC(<CADENA DE CARACTERES>)	——	ATN(<X>)	CHR$(<NUMERO DE CODIGO ASCII>)
MACINTOSH	ABS(<NUMERO>)	——	ASC(<CADENA DE CARACTERES>)	——	ATN(<X>)	CHR$(<NUMERO DE CODIGO ASCII>)

	CLOAD ["<NOMBRE>"]	CLOSE#<NUMERO DE CANAL>	COS(<X>)
APPLE	——	CLOSE <NOMBRE DE FICHERO>	COS(<X>)
COMMODORE 64	——	CLOSE#<NUMERO DE CANAL>[,#<NUMERO DE CANAL>,...]	COS(<X>)
MSX	CLOAD ["<NOMBRE>"]	CLOSE#<NUMERO DE CANAL>[,#<NUMERO DE CANAL>,...]	COS(<X>)
IBM - PC	——	CLOSE#<NUMERO DE CANAL>[,#<NUMERO DE CANAL>,...]	COS(<X>)
QL	CLOAD ["<NOMBRE>"]	CLOSE#<NUMERO DE CANAL>[,#<NUMERO DE CANAL>,...]	COS(<X>)
SPECTRAVIDEO	CLOAD ["<NOMBRE>"]	CLOSE#<NUMERO DE CANAL>[,#<NUMERO DE CANAL>,...]	COS(<X>)
SPECTRUM	——	——	COS(<X>)
AMSTRAD 664	CLOAD ["<NOMBRE>"]	CLOSEOUT	COS(<X>)
MACINTOSH	——	CLOSE#<NUMERO DE CANAL>[,#<NUMERO DE CANAL>,...]	COS(<X>)

	CSAVE ["<NOMBRE>"]	CVD(<VARIABLE DE CADENA>)	CVI(<VARIABLE DE CADENA>)	CVS (<VARIABLE DE CADENA>)
APPLE	——	——	——	——
COMMODORE 64	——	——	——	——
MSX	CSAVE ["<NOMBRE>"]	CVD(<VARIABLE DE CADENA>)	CVI(<VARIABLE DE CADENA>)	CVS (<VARIABLE DE CADENA>)
IBM - PC	——	CVD(<VARIABLE DE CADENA>)	CVI(<VARIABLE DE CADENA>)	CVS (<VARIABLE DE CADENA>)
QL	CSAVE ["<NOMBRE>"]	——	——	——
SPECTRAVIDEO	CSAVE ["<NOMBRE>"]	——	——	——
SPECTRUM	——	——	——	——
AMSTRAD 664	CSAVE ["<NOMBRE>"]	——	——	——
MACINTOSH	——	CVD(<VARIABLE DE CADENA>)	CVI(<VARIABLE DE CADENA>)	CVS (<VARIABLE DE CADENA>)

	DATA <CONSTANTE1>[,<CONSTANTE2>,...]	DEF FN<NOMBRE DE LA FUNCION>(<PARAMETRO>)=<EXPRESION DE CADENA>
APPLE	DATA <CONSTANTE1>[,<CONSTANTE2>,...]	——
COMMODORE 64	DATA <CONSTANTE1>[,<CONSTANTE2>,...]	——
MSX	DATA <CONSTANTE1>[,<CONSTANTE2>,...]	DEF FN<NOMBRE DE LA FUNCION>(<PARAMETRO>)=<EXPRESION DE CADENA>
IBM - PC	DATA <CONSTANTE1>[,<CONSTANTE2>,...]	DEF FN<NOMBRE DE LA FUNCION>(<PARAMETRO>)=<EXPRESION DE CADENA>
QL	DATA <CONSTANTE1>[,<CONSTANTE2>,...]	DEF FN<NOMBRE DE LA FUNCION>(<PARAMETRO>)=<EXPRESION DE CADENA>
SPECTRAVIDEO	DATA <CONSTANTE1>[,<CONSTANTE2>,...]	DEF FN<NOMBRE DE LA FUNCION>(<PARAMETRO>)=<EXPRESION DE CADENA>
SPECTRUM	DATA <CONSTANTE1>[,<CONSTANTE2>,...]	DEF FN<NOMBRE DE LA FUNCION>(<PARAMETRO>)=<EXPRESION DE CADENA>
AMSTRAD 664	DATA <CONSTANTE1>[,<CONSTANTE2>,...]	——
MACINTOSH	DATA <CONSTANTE1>[,<CONSTANTE2>,...]	——

	DIM(<NUMERO DE FILAS,NUMERO DE COLUMNAS>)	END
APPLE	DIM(<NUMERO DE FILAS,NUMERO DE COLUMNAS>)	END
COMMODORE 64	DIM(<NUMERO DE FILAS,NUMERO DE COLUMNAS>)	END
MSX	DIM(<NUMERO DE FILAS,NUMERO DE COLUMNAS>)	END
IBM - PC	DIM(<NUMERO DE FILAS,NUMERO DE COLUMNAS>)	END
QL	DIM(<NUMERO DE FILAS,NUMERO DE COLUMNAS>)	END
SPECTRAVIDEO	DIM(<NUMERO DE FILAS,NUMERO DE COLUMNAS>)	END
SPECTRUM	DIM(<NUMERO DE FILAS,NUMERO DE COLUMNAS>)	END
AMSTRAD 664	DIM(<NUMERO DE FILAS,NUMERO DE COLUMNAS>)	END
MACINTOSH	DIM(<NUMERO DE FILAS,NUMERO DE COLUMNAS>)	END

	FIELD#<NUMERO DE CANAL>,<LONGITUD1> AS <VARIABLE DE CADENA1>[,<LONGITUD2> AS <VARIABLE DE CADENA2>,..]
APPLE	——
COMMODORE 64	——
MSX	FIELD#<NUMERO DE CANAL>,<LONGITUD1> AS <VARIABLE DE CADENA1>[,<LONGITUD2> AS <VARIABLE DE CADENA2>,..]
IBM - PC	FIELD#<NUMERO DE CANAL>,<LONGITUD1> AS <VARIABLE DE CADENA1>[,<LONGITUD2> AS <VARIABLE DE CADENA2>,..]
QL	FIELD#<NUMERO DE CANAL>,<LONGITUD1> AS <VARIABLE DE CADENA1>[,<LONGITUD2> AS <VARIABLE DE CADENA2>,..]
SPECTRAVIDEO	FIELD#<NUMERO DE CANAL>,<LONGITUD1> AS <VARIABLE DE CADENA1>[,<LONGITUD2> AS <VARIABLE DE CADENA2>,..]
SPECTRUM	——
AMSTRAD 664	——
MACINTOSH	FIELD#<NUMERO DE CANAL>,<LONGITUD1> AS <VARIABLE DE CADENA1>[,<LONGITUD2> AS <VARIABLE DE CADENA2>,..]

	FOR <VARIABLE>=<INICIAL> TO <FINAL> STEP <SALTO>	NEXT <VARIABLE>	GET#
APPLE	FOR <VARIABLE>=<INICIAL> TO <FINAL> STEP <SALTO>	NEXT <VARIABLE>	GET <VARIABLE> (*)
COMMODORE 64	FOR <VARIABLE>=<INICIAL> TO <FINAL> STEP <SALTO>	NEXT <VARIABLE>	GET#<N.DE CANAL>,<VARIABLE> (**)
MSX	FOR <VARIABLE>=<INICIAL> TO <FINAL> STEP <SALTO>	NEXT <VARIABLE>	———
IBM - PC	FOR <VARIABLE>=<INICIAL> TO <FINAL> STEP <SALTO>	NEXT <VARIABLE>	GET#<N.DE CANAL>,<N.DE REGISTRO>
QL	FOR <VARIABLE>=<INICIAL> TO <FINAL> STEP <SALTO>	NEXT <VARIABLE>	———
SPECTRAVIDEO	FOR <VARIABLE>=<INICIAL> TO <FINAL> STEP <SALTO>	NEXT <VARIABLE>	GET#<N.DE CANAL>,<N.DE REGISTRO>
SPECTRUM	FOR <VARIABLE>=<INICIAL> TO <FINAL> STEP <SALTO>	NEXT <VARIABLE>	———
AMSTRAD 664	FOR <VARIABLE>=<INICIAL> TO <FINAL> STEP <SALTO>	NEXT <VARIABLE>	———
MACINTOSH	FOR <VARIABLE>=<INICIAL> TO <FINAL> STEP <SALTO>	NEXT <VARIABLE>	GET#<N.DE CANAL>,<N.DE REGISTRO>

(*) En el Apple II los comandos de manejo de caracteres, precedidos de CTRL-D.
(**) En el Commodore 64 el GET# toma un solo carácter cada vez y se lo asigna a la variable indicada.

	GOSUB <NUMERO DE LINEA>	GOTO<NUMERO DE LINEA>	GOTO<VARIABLE>	IF/THEN[/ELSE]	INPUT <VARIABLE>
APPLE	GOSUB <NUMERO DE LINEA>	GOTO<NUMERO DE LINEA>	GOTO<VARIABLE>	IF/THEN[/ELSE]	INPUT <VARIABLE>
COMMODORE 64	GOSUB <NUMERO DE LINEA>	GOTO<NUMERO DE LINEA>	GOTO<VARIABLE>	IF/THEN[/ELSE]	INPUT <VARIABLE>
MSX	GOSUB <NUMERO DE LINEA>	GOTO<NUMERO DE LINEA>	GOTO<VARIABLE>	IF/THEN[/ELSE]	INPUT <VARIABLE>
IBM - PC	GOSUB <NUMERO DE LINEA>	GOTO<NUMERO DE LINEA>	GOTO<VARIABLE>	IF/THEN[/ELSE]	INPUT <VARIABLE>
QL	GOSUB <NUMERO DE LINEA>	GOTO<NUMERO DE LINEA>	GOTO<VARIABLE>	IF/THEN[/ELSE]	INPUT <VARIABLE>
SPECTRAVIDEO	GOSUB <NUMERO DE LINEA>	GOTO<NUMERO DE LINEA>	GOTO<VARIABLE>	IF/THEN[/ELSE]	INPUT <VARIABLE>
SPECTRUM	GOSUB <NUMERO DE LINEA>	GOTO<NUMERO DE LINEA>	GOTO<VARIABLE>	IF/THEN	INPUT <VARIABLE>
AMSTRAD 664	GOSUB <NUMERO DE LINEA>	GOTO<NUMERO DE LINEA>	GOTO<VARIABLE>	IF/THEN[/ELSE]	INPUT <VARIABLE>
MACINTOSH	GOSUB <NUMERO DE LINEA>	GOTO<NUMERO DE LINEA>	GOTO<VARIABLE>	IF/THEN[/ELSE]	INPUT <VARIABLE>

	INPUT "<MENSAJE>"{;,}<VARIABLE>	INPUT#	INT(<NUMERO>)
APPLE	INPUT "<MENSAJE>"{;,}<VARIABLE>	INPUT "<MENSAJE>"{,;}<VARIABLE>[,<VARIABLE2>,...]	INT(<NUMERO>)
COMMODORE 64	INPUT "<MENSAJE>"{;,}<VARIABLE>	INPUT "<MENSAJE>"{,;}<VARIABLE>[,<VARIABLE2>,...]	INT(<NUMERO>)
MSX	INPUT "<MENSAJE>"{;,}<VARIABLE>	INPUT #<N.DE CANAL>,<VARIABLE1>[,<VARIABLE2>,...]	INT(<NUMERO>)
IBM - PC	INPUT "<MENSAJE>"{;,}<VARIABLE>	INPUT #<N.DE CANAL>,<VARIABLE1>[,<VARIABLE2>,...]	INT(<NUMERO>)
QL	INPUT "<MENSAJE>"{;,}<VARIABLE>	INPUT #<N.DE CANAL>,<VARIABLE1>[,<VARIABLE2>,...]	INT(<NUMERO>)
SPECTRAVIDEO	INPUT "<MENSAJE>"{;,}<VARIABLE>	INPUT #<N.DE CANAL>,<VARIABLE1>[,<VARIABLE2>,...]	INT(<NUMERO>)
SPECTRUM	INPUT "<MENSAJE>"{;,}<VARIABLE>	———	INT(<NUMERO>)
AMSTRAD 664	INPUT "<MENSAJE>"{;,}<VARIABLE>	INPUT #<N.DE CANAL>,<VARIABLE1>[,<VARIABLE2>,...]	INT(<NUMERO>)
MACINTOSH	INPUT "<MENSAJE>"{;,}<VARIABLE>	INPUT #<N.DE CANAL>,<VARIABLE1>[,<VARIABLE2>,...]	INT(<NUMERO>)

	LEFT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	LEN (<CADENA DE CARACTERES>)
APPLE	LEFT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	LEN (<CADENA DE CARACTERES>)
COMMODORE 64	LEFT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	LEN (<CADENA DE CARACTERES>)
MSX	LEFT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	LEN (<CADENA DE CARACTERES>)
IBM - PC	LEFT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	LEN (<CADENA DE CARACTERES>)
QL	———	LEN (<CADENA DE CARACTERES>)
SPECTRAVIDEO	LEFT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	LEN (<CADENA DE CARACTERES>)
SPECTRUM	———	LEN (<CADENA DE CARACTERES>)
AMSTRAD 664	LEFT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	LEN (<CADENA DE CARACTERES>)
MACINTOSH	LEFT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	LEN (<CADENA DE CARACTERES>)

	LET<VARIABLE>=<EXPRESION>	<VARIABLE>=<EXPRESION>	LINE INPUT#
APPLE	LET<VARIABLE>=<EXPRESION>	<VARIABLE>=<EXPRESION>	———
COMMODORE 64	LET<VARIABLE>=<EXPRESION>	<VARIABLE>=<EXPRESION>	———
MSX	LET<VARIABLE>=<EXPRESION>	<VARIABLE>=<EXPRESION>	LINE INPUT #<N.DE CANAL>,<VARIABLE1>[,<VARIABLE2>,...]
IBM - PC	LET<VARIABLE>=<EXPRESION>	<VARIABLE>=<EXPRESION>	LINE INPUT #<N.DE CANAL>,<VARIABLE1>[,<VARIABLE2>,...]
QL	LET<VARIABLE>=<EXPRESION>	<VARIABLE>=<EXPRESION>	———
SPECTRAVIDEO	LET<VARIABLE>=<EXPRESION>	<VARIABLE>=<EXPRESION>	LINE INPUT #<N.DE CANAL>,<VARIABLE1>[,<VARIABLE2>,...]
SPECTRUM	LET<VARIABLE>=<EXPRESION>	———	———
AMSTRAD 664	LET<VARIABLE>=<EXPRESION>	<VARIABLE>=<EXPRESION>	LINE INPUT #<N.DE CANAL>,<VARIABLE1>[,<VARIABLE2>,...]
MACINTOSH	LET<VARIABLE>=<EXPRESION>	<VARIABLE>=<EXPRESION>	LINE INPUT #<N.DE CANAL>,<VARIABLE1>[,<VARIABLE2>,...]

	LOAD	LOF	LSET <VARIABLE DE CAMPO>=<VARIABLE>
APPLE	LOAD "<NOMBRE>"	———	———
COMMODORE 64	LOAD "<NOMBRE>",<PERIFERICO>	———	———
MSX	LOAD "<PERIFERICO>:<NOMBRE>"	———	LSET <VARIABLE DE CAMPO>=<VARIABLE>
IBM - PC	LOAD "<PERIFERICO>:<NOMBRE>"	LOF(<NUMERO DE CANAL>)	LSET <VARIABLE DE CAMPO>=<VARIABLE>
QL	LOAD "<PERIFERICO>_<NOMBRE>"	———	———
SPECTRAVIDEO	LOAD "<PERIFERICO>:<NOMBRE>"	———	LSET <VARIABLE DE CAMPO>=<VARIABLE>
SPECTRUM	LOAD "<NOMBRE>"	———	———
AMSTRAD 664	LOAD "<NOMBRE>"	———	———
MACINTOSH	LOAD "<NOMBRE>"	LOF(<NUMERO DE CANAL>)	LSET <VARIABLE DE CAMPO>=<VARIABLE>

	MID$(<CADENA>,<PRIMER CARACTER A EXTRAER>,<NUMERO DE CARACTERES A EXTRAER>)	MKD$(<VARIABLE>)	MKI$(<VARIABLE>)
APPLE	MID$(<CADENA>,<PRIMER CARACTER A EXTRAER>,<NUMERO DE CARACTERES A EXTRAER>)	———	———
COMMODORE 64	MID$(<CADENA>,<PRIMER CARACTER A EXTRAER>,<NUMERO DE CARACTERES A EXTRAER>)		———
MSX	MID$(<CADENA>,<PRIMER CARACTER A EXTRAER>,<NUMERO DE CARACTERES A EXTRAER>)	MKD$(<VARIABLE>)	MKI$(<VARIABLE>)
IBM - PC	MID$(<CADENA>,<PRIMER CARACTER A EXTRAER>,<NUMERO DE CARACTERES A EXTRAER>)	MKD$(<VARIABLE>)	MKI$(<VARIABLE>)
QL	———	———	———
SPECTRAVIDEO	MID$(<CADENA>,<PRIMER CARACTER A EXTRAER>,<NUMERO DE CARACTERES A EXTRAER>)	———	———
SPECTRUM	———	———	———
AMSTRAD 664	MID$(<CADENA>,<PRIMER CARACTER A EXTRAER>,<NUMERO DE CARACTERES A EXTRAER>)	———	———
MACINTOSH	MID$(<CADENA>,<PRIMER CARACTER A EXTRAER>,<NUMERO DE CARACTERES A EXTRAER>)	MKD$(<VARIABLE>)	MKI$(<VARIABLE>)

	MKS$(<VARIABLE>)	ON ERROR GOTO <NUMERO DE LINEA>	ON ERROR GOSUB <NUMERO DE LINEA>
APPLE	———	ONERR GOTO <NUMERO DE LINEA>	ONERR GOSUB <NUMERO DE LINEA>
COMMODORE 64	———	———	———
MSX	MKS$(<VARIABLE>)	ON ERROR GOTO <NUMERO DE LINEA>	ON ERROR GOSUB <NUMERO DE LINEA>
IBM - PC	MKS$(<VARIABLE>)	ON ERROR GOTO <NUMERO DE LINEA>	ON ERROR GOSUB <NUMERO DE LINEA>
QL	———	———	———
SPECTRAVIDEO	———	ON ERROR GOTO <NUMERO DE LINEA>	ON ERROR GOSUB <NUMERO DE LINEA>
SPECTRUM	———	———	———
AMSTRAD 664	———	ON ERROR GOTO <NUMERO DE LINEA>	ON ERROR GOSUB <NUMERO DE LINEA>
MACINTOSH	MKS$(<VARIABLE>)	ON ERROR GOTO <NUMERO DE LINEA>	ON ERROR GOSUB <NUMERO DE LINEA>

	ON <EXPRESION> GOSUB <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
APPLE	ON <EXPRESION> GOSUB <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
COMMODORE 64	ON <EXPRESION> GOSUB <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
MSX	ON <EXPRESION> GOSUB <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
IBM - PC	ON <EXPRESION> GOSUB <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
QL	ON <EXPRESION> GOSUB <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
SPECTRAVIDEO	ON <EXPRESION> GOSUB <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
SPECTRUM	———
AMSTRAD 664	ON <EXPRESION> GOSUB <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
MACINTOSH	ON <EXPRESION> GOSUB <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]

	ON <EXPRESION> GOTO <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
APPLE	ON <EXPRESION> GOTO <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
COMMODORE 64	ON <EXPRESION> GOTO <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
MSX	ON <EXPRESION> GOTO <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
IBM - PC	ON <EXPRESION> GOTO <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
QL	ON <EXPRESION> GOTO <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
SPECTRAVIDEO	ON <EXPRESION> GOTO <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
SPECTRUM	———
AMSTRAD 664	ON <EXPRESION> GOTO <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]
MACINTOSH	ON <EXPRESION> GOTO <NUMERO DE LINEA1>[,<NUMERO DE LINEA2>,..]

	OPEN (Para ficheros de acceso directo)
APPLE	OPEN "<NOMBRE DE FICHERO>",Ln1,Sn2,Dn2,Vn4 (*)
COMMODORE 64	———
MSX	———
IBM - PC	OPEN "R",#<N.CANAL>,"<NOMBRE DE FICHERO>",<LONG. REGISTRO>
QL	———
SPECTRAVIDEO	———
SPECTRUM	———
AMSTRAD 664	———
MACINTOSH	OPEN "R",#<N.CANAL>,"<NOMBRE>",<LONG.REGISTRO>

(*) En el Apple II los comandos de manejo de ficheros se utilizan como cadenas de caracteres, precedidos de CTRL-D.

	OPEN (Para ficheros secuenciales)
APPLE	OPEN <NOMBRE DE FICHERO> (*)
COMMODORE 64	OPEN "<MODO>",#<NUMERO DE CANAL>,"<NOMBRE DE FICHERO>"
MSX	OPEN "[<DISPOSITIVO>:]<NOMBRE DE FICHERO>"FOR <MODO> AS #<NUMERO DE CANAL>
IBM - PC	OPEN "<MODO>",#<NUMERO DE CANAL>,"<NOMBRE DE FICHERO>"
QL	OPEN#<NUMERO DE CANAL>,"<NOMBRE DE FICHERO>"
SPECTRAVIDEO	OPEN "[<DISPOSITIVO>:]<NOMBRE DE FICHERO>"FOR <MODO> AS #<NUMERO DE CANAL>
SPECTRUM	———
AMSTRAD 664	OPENOUT "<NOMBRE DE REGISTRO>"
MACINTOSH	OPEN "<MODO>",#<NUMERO DE CANAL>,"<NOMBRE DE FICHERO>"

(*) En el Apple II los comandos de manejo de ficheros se utilizan como cadenas de caracteres, precedidos de CTRL-D.

	PRINT	PRINT,	PRINT;	PRINT "<MENSAJE>"	PRINT X	PRINT "<VARIABLE>"	PRINT#<NUMERO DE CANAL>,<DATO>
APPLE	PRINT	PRINT,	PRINT;	PRINT "<MENSAJE>"	PRINT X	PRINT "<VARIABLE>"	WRITE <NOMBRE>
COMMODORE 64	PRINT	PRINT,	PRINT;	PRINT "<MENSAJE>"	PRINT X	PRINT "<VARIABLE>"	PRINT#<NUMERO DE CANAL>,<DATO>
MSX	PRINT	PRINT,	PRINT;	PRINT "<MENSAJE>"	PRINT X	PRINT "<VARIABLE>"	PRINT#<NUMERO DE CANAL>,<DATO>
IBM - PC	PRINT	PRINT,	PRINT;	PRINT "<MENSAJE>"	PRINT X	PRINT "<VARIABLE>"	PRINT#<NUMERO DE CANAL>,<DATO>
QL	PRINT	PRINT,	PRINT;	PRINT "<MENSAJE>"	PRINT X	PRINT "<VARIABLE>"	PRINT#<NUMERO DE CANAL>,<DATO>
SPECTRAVIDEO	PRINT	PRINT,	PRINT;	PRINT "<MENSAJE>"	PRINT X	PRINT "<VARIABLE>"	PRINT#<NUMERO DE CANAL>,<DATO>
SPECTRUM	PRINT	PRINT,	PRINT;	PRINT "<MENSAJE>"	PRINT X	PRINT "<VARIABLE>"	SAVE "<NOMBRE DE PROGRAMA>" DATA <ARRAY>
AMSTRAD 664	PRINT	PRINT,	PRINT;	PRINT "<MENSAJE>"	PRINT X	PRINT "<VARIABLE>"	WRITE#<NUMERO DE CANAL>,<DATO1>[,<DATO2>,...]
MACINTOSH	PRINT	PRINT,	PRINT;	PRINT "<MENSAJE>"	PRINT X	PRINT "<VARIABLE>"	WRITE#<NUMERO DE CANAL>,<DATO1>[,<DATO2>,...]

	PRINT#	PUT#<N.DE CANAL>,<N.DE REGISTRO>
APPLE	PRINT "<MENSAJE>"{,;}<VARIABLE1>[,<VARIABLE2>,...]	
COMMODORE 64	PRINT #,"<MENSAJE>"{,;}<VARIABLE1>[,<VARIABLE2>,...]	———
MSX	PRINT #<NUMERO DE CANAL>,"<MENSAJE>"{,;}<VARIABLE1>[,<VARIABLE2>,...]	———
IBM - PC	PRINT #<NUMERO DE CANAL>,"<MENSAJE>"{,;}<VARIABLE1>[,<VARIABLE2>,...]	———
QL	PRINT #<NUMERO DE CANAL>,"<MENSAJE>"{,;}<VARIABLE1>[,<VARIABLE2>,...]	PUT#<N.DE CANAL>,<N.DE REGISTRO>
SPECTRAVIDEO	PRINT #<NUMERO DE CANAL>,"<MENSAJE>"{,;}<VARIABLE1>[,<VARIABLE2>,...]	———
SPECTRUM	———	———
AMSTRAD 664	———	———
MACINTOSH	PRINT #<NUMERO DE CANAL>,"<MENSAJE>"{,;}<VARIABLE1>[,<VARIABLE2>,...]	PUT#<N.DE CANAL>,<N.DE REGISTRO>

	READ <VARIABLE1>[,<VARIABLE2>,...]	RESTORE [<NUMERO DE LINEA>]	RETURN
APPLE	READ <VARIABLE1>[,<VARIABLE2>,...]	RESTORE	RETURN
COMMODORE 64	READ <VARIABLE1>[,<VARIABLE2>,...]	RESTORE	RETURN
MSX	READ <VARIABLE1>[,<VARIABLE2>,...]	RESTORE [<NUMERO DE LINEA>]	RETURN
IBM - PC	READ <VARIABLE1>[,<VARIABLE2>,...]	RESTORE [<NUMERO DE LINEA>]	RETURN
QL	READ <VARIABLE1>[,<VARIABLE2>,...]	RESTORE [<NUMERO DE LINEA>]	RETURN
SPECTRAVIDEO	READ <VARIABLE1>[,<VARIABLE2>,...]	RESTORE [<NUMERO DE LINEA>]	RETURN
SPECTRUM	READ <VARIABLE1>[,<VARIABLE2>,...]	RESTORE	RETURN
AMSTRAD 664	READ <VARIABLE1>[,<VARIABLE2>,...]	RESTORE [<NUMERO DE LINEA>]	RETURN
MACINTOSH	READ <VARIABLE1>[,<VARIABLE2>,...]	RESTORE [<NUMERO DE LINEA>]	RETURN

	RIGHT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	RND[<NUMERO POSITIVO>]	RSET <VARIABLE DE CAMPO>=<VARIABLE>
APPLE	RIGHT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	RND[<NUMERO POSITIVO>]	———
COMMODORE 64	RIGHT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	RND[<NUMERO POSITIVO>]	———
MSX	RIGHT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	RND[<NUMERO POSITIVO>]	RSET <VARIABLE DE CAMPO>=<VARIABLE>
IBM - PC	RIGHT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	RND[<NUMERO POSITIVO>]	RSET <VARIABLE DE CAMPO>=<VARIABLE>
QL	———	RND[<NUMERO POSITIVO>]	———
SPECTRAVIDEO	RIGHT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	RND[<NUMERO POSITIVO>]	RSET <VARIABLE DE CAMPO>=<VARIABLE>
SPECTRUM	———	RND[<NUMERO POSITIVO>]	———
AMSTRAD 664	RIGHT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	RND[<NUMERO POSITIVO>]	———
MACINTOSH	RIGHT$(<CADENA>,<NUMERO DE CARACTERES A EXTRAER>)	RANDOM[<NUMERO POSITIVO>]	RSET <VARIABLE DE CAMPO>=<VARIABLE>

	RUN	RUN <NUMERO DE LINEA>	SAVE
APPLE	RUN	RUN <NUMERO DE LINEA>	SAVE "<NOMBRE>"
COMMODORE 64	RUN	RUN <NUMERO DE LINEA>	SAVE "<NOMBRE>",<PERIFERICO>
MSX	RUN	RUN <NUMERO DE LINEA>	SAVE "<PERIFERICO>:<NOMBRE>"
IBM - PC	RUN	RUN <NUMERO DE LINEA>	SAVE "<PERIFERICO>:<NOMBRE>"
QL	RUN	RUN <NUMERO DE LINEA>	SAVE "<PERIFERICO>_<NOMBRE>"
SPECTRAVIDEO	RUN	RUN <NUMERO DE LINEA>	SAVE "<PERIFERICO>:<NOMBRE>"
SPECTRUM	RUN	RUN <NUMERO DE LINEA>	SAVE "<NOMBRE>"
AMSTRAD 664	RUN	RUN <NUMERO DE LINEA>	SAVE "<NOMBRE>"
MACINTOSH	RUN	RUN <NUMERO DE LINEA>	SAVE "<NOMBRE>"

	SIN(<X>)	SQR(<X>)	TAN(<X>)	VERIFY "<NOMBRE>"
APPLE	SIN(<X>)	SQR(<X>)	TAN(<X>)	VERIFY "<NOMBRE>"
COMMODORE 64	SIN(<X>)	SQR(<X>)	TAN(<X>)	VERIFY "<NOMBRE>"[,"<PERIFERICO>"]
MSX	SIN(<X>)	SQR(<X>)	TAN(<X>)	CLOAD? "<NOMBRE>"
IBM - PC	SIN(<X>)	SQR(<X>)	TAN(<X>)	———
QL	SIN(<X>)	SQR(<X>)	TAN(<X>)	———
SPECTRAVIDEO	SIN(<X>)	SQR(<X>)	TAN(<X>)	CLOAD? "<NOMBRE>"
SPECTRUM	SIN(<X>)	SQR(<X>)	TAN(<X>)	VERIFY "<NOMBRE>"
AMSTRAD 664	SIN(<X>)	SQR(<X>)	TAN(<X>)	———
MACINTOSH	SIN(<X>)	SQR(<X>)	TAN(<X>)	———

BIBLIOGRAFIA

Libros recomendados para el BASIC, la informática y los ordenadores personales

Esperamos que la lectura de las dos monografías relativas al BASIC en general hayan conseguido el objetivo que pretendían: proporcionarles un cierto conocimiento de este lenguaje y, quizá, animarles a escribir por sí mismos algún pequeño programa. Si, por el contrario, no se siente todavía con el dominio suficiente para desarrollar un programa, al menos le será más fácil comprender los listados de los numerosos programas que encontrará publicados en las revistas especializadas o de los cuales tenga una copia en cinta.

Estamos convencidos, no obstante, de que su interés por la informática de los ordenadores personales está solamente en sus comienzos y querrá saber mucho más. Estas monografías satisfacen dicha necesidad y, sin un compromiso de lectura profunda, podrá encontrar en ellas un poco de todo lo que es actualidad en los ordenadores personales.

Si, por el contrario, hemos despertado en usted un mayor interés por el BASIC, entonces le aconsejamos leer otras obras que, con mayor dedicación, profundizan mejor en las materias y situaciones diversas. Una recomendación que siempre repetimos es la de leer con atención los manuales que se suministran con los ordenadores. Somos conscientes de que a menudo están demasiado resumidos, redactados en inglés o, peor todavía, mal traducidos. Pero al menos en dichos manuales encontrará las instrucciones específicas del ordenador que utiliza.

A continuación deseamos facilitarle una serie de obras interesantes que se refieren a casi todos los fundamentos de la informática (nos disculpamos si algunos están con su título en inglés) y también indicamos algunos textos de fácil consulta que se refieren a la programación en BASIC de ordenadores particulares.

Programación en BASIC: un método práctico.
Dachslager y Zucker. *Anaya Multimedia.*

Diseño de gráficos y videojuegos. Tratamiento en tres dimensiones.
Angel y Jones, 1985. *Anaya Multimedia.*

Programación avanzada en BASIC.
Bishop. *Anaya Multimedia.*

El libro del IBM, PC, XT, AT.
L. E. Frenzel Jr. / L. E. Frenzel III, 1985. *Anaya Multimedia.*

Diccionario de informática ingles-español-francés.
G. A. Mania, 1985. *Paraninfo.*

Diccionario del BASIC.
Willie Hart, 1985. *Paraninfo.*

Como programar su COMMODORE 64 1 - BASIC, gráficos, sonido.
F. Montell, 1985. *Paraninfo.*

Como programar un COMMODORE 64 2 - Lenguaje máquina, E/S, periféricos.
F. Montell, 1985. *Paraninfo.*

Tratamientos de textos con BASIC.
G. Quaneaux, 1985. *Paraninfo.*

Informática para no avanzados.
G. Willmott, 1985. *Deusto.*

102 programas para su APPLE.
Desconchat. *Elisa.*

Claves para el COMMODORE 64.
D. Gean David. *Elisa.*

COMMODORE 64, para todos.
Boisgontier Brebion Foucault. *Elisa.*

Los ordenadores. Fundamentos y sistemas.
J. C. Giarratano, 1984. *Díaz de Santos.*

Conceptos actuales sobre la tecnología de los ordenadores.
J. C. Giarratano, 1984. *Díaz de Santos.*

Diccionario de informática ingles-español. Glosario de términos informáticos.
Olivetti, 5. Edic, 1984. *Paraninfo.*

Como prograrmar ordenadores personales.
R. Farrando, 1985. *Marcombo.*

Diccionario de informática.
Masson, 2. Edic, 1985. *Masson.*

Glosario de computación.
Alan Freedman, 1984. *McGraw-Hill.*

Diccionario del BASIC.
A. Lien. *Ed. Elisa.*

Claves para el Appe II.
Breaud-Pouliquen. *Ed. Elisa.*

Claves para el Commodore 64.
D. J. David. *Ed. Elisa.*

Claves para el ZX-Spectrum.
J. Francois Séhan. *Ed. Elisa.*

El BASIC de la A a la Z.
J. Boisgontrer. *Ed. Elisa.*

Commodore 64 para todos.
J. Boisgontrer. *Ed. Elisa*

102 Programas para Commodore 64.
J. Deconchat. *Ed. Elisa.*

102 Programas para ZX81 y Spectrum.
J. Deconchat. *Ed. Elisa.*

El Apple y sus ficheros.
J. Boisgontrer. *Ed. Elisa.*

Microordenadores y cassettes.
M. Salem. *Ed. Noray.*

Profundizando en el ZX-Spectrum.
D. Jones. *Ed. Noray.*

BIBLIOTECA BASICA INFORMATICA

INDICE GENERAL

10 Sistemas operativos y software de base
Qué son, para qué sirven. Unos desconocidos muy importantes.

11 Sistema operativo CP/M
Uno de los sistemas operativos para microprocesadores de 8 bits de mayor difusión en el mercado.

12 MS-DOS: el estándar de IBM
Sistema operativo para el microprocesador de 16 bits 8088, adoptado por el IBM-PC.

13 Paquetes de aplicaciones. Software "pret a porter"
Características y peculiaridades de los más importantes paquetes de aplicaciones.

14 VisiCalc: una buena hoja de cálculo
Interioridades y manejo de una de las hojas de cálculo más usadas.

15 Dibujar con el ordenador
Profundizando en una de las facetas útiles y divertidas que nos ofrecen los ordenadores.

16 Tratamiento de textos... para escribir con el ordenador
Cómo convertir su ordenador en una máquina de escribir con memoria y todo tipo de posibilidades.

17 Diseño de juegos
Particularidades características de esta aplicación de los ordenadores.

18 LOGO: la tortuga inteligente
Un lenguaje conocido por su «cursor gráfico», la tortuga, y sus aplicaciones pedagógicas al alcance de su mano.

19 BASIC y tratamiento de imágenes
Todo lo que en **¡Y llegó el BASIC!** no se pudo ver sobre las imágenes y gráficos en el BASIC.

20 Bancos de datos (I)
Peculiaridades de una de las aplicaciones de los ordenadores más interesantes, y que más dinero mueven.

21 Bancos de datos (II)
Profundizando en sus características.

22 Paquetes integrados: Lotus 1-2-3 y Simphony
Estudio de dos de los paquetes integrados (Hoja de cálculo + base de datos + ...) más conocidos.

23 dBASE II y dBASE III
Cómo aprovechar las dos versiones más recientes de esta importante base de datos.

24 Los ordenadores uno a uno
Un amplio y completo estudio comparativo.

25 Cálculo numérico en BASIC
Una aplicación especializada a su disposición.

26 Multiplan
Cómo hacer uso de este moderno paquete de aplicaciones.

27 FORTRAN y COBOL
Dos lenguajes muy especializados y distintos.

28 FORTH: anatomía de un lenguaje inteligente
Principales características de un lenguaje moderno, flexible y de amplio uso, en la robótica.

29 Cómo realizar nuestro propio banco de datos
Conocimientos necesarios para poder fabricar un banco de datos a nuestro gusto y medida.

30 Los paquetes integrados uno a uno
Todos los que usted puede encontrar en el mercado.

NOTA: Ingelek, S. A. se reserva el derecho de modificar, sin previo aviso, el orden, título o contenido de cualquier volumen de la colección.

NOTAS

Tenemos la seguridad de que muchos de nuestros lectores estaban esperando ansiosos este segundo volumen dedicado al BASIC "de andar por casa" de nuestra colección. Efectivamente, en "Y llegó el BASIC... I" quedaron muchos temas por tratar: subrutinas, funciones, instrucciones de salto, ficheros, etc., que ahora vamos a estudiar. También incluimos una tabla de equivalencias de las instrucciones vistas para distintos ordenadores.

Dado lo amplio de la temática irán apareciendo sucesivamente otros volúmenes de la B. B. I. dedicados a profundizar en muchos de los aspectos que han aparecido en los tomos I y II de "Y llegó el BASIC".

375 pts.